AKTUALA KOMPUTIKO

(nur por komputistoj)

NUMERO 1 MARTO 1985

Numero 1 Marto 1985

AKTUALA KOMPUTIKO

(nur por komputistoj)

Redaktoro :

Christian BERTIN
CCETT AIS
B.P : 59
Rue du Clos Courtel
F–35510 CESSON–SEVIGNE
FRANCIO

Aktuala Komputiko
Nur por komputistoj
=======

Esperanto :

"Aktuala Komputiko" estas internacia fakrevuo pri nuntempaj esploroj en la diversaj fakoj rilataj al komputiloj kaj uzo de komputiloj.

En "Aktuala Komputiko" aperas nur originalaj artikoloj verkitaj de fakuloj, kaj ĉiam kun almenaŭ unu resumo en nacia lingvo.

Kial esperantlingva fakrevuo ?

- Pro la fakto ke la scienco kaj Esperanto estas internaciaj kaj neŭtralaj, tio signifas ke ili ne apartenas al iu ajn bloko ĉu lingva aŭ rasa, ĉu okcidenta aŭ orienta, ĉu norda aŭ suda, ĉu de antaŭevoluantaj aŭ de postevoluantaj landoj, sed ambaŭ scienco kaj Esperanto apartenas al la tuta homaro.

Se ankaŭ vi konscias aparteni al universala komunumo kaj opinias ke la scienco devas progresi por la feliĉo kaj la prospero de ĉiuj popoloj, tiam por pliproksimigi la sciencon al ĉiuj popoloj, plej bone estas verki en la plej facile lernebla lingvo : Esperanto, fremda lingvo por ĉiuj homoj, ĝi ankaŭ forigas ĉiun lingvan diskriminacion devigante ĉiujn terloĝantojn lerni almenaŭ unu fremdan lingvon por la internacia komunikado.

La redaktoro.

P.S : En via lando ekzistas asocio por Esperanto kiu pretas informi vin se necesas, ĝian adreson vi trovos fine de tiu numero.

English :

"Aktuala Komputiko" is an international computer journal concerned with current research in various fields related to computers and their usage.

Only original articles written by specialists appear in "Aktuala Komputiko" always with at least one résumé in a national language.

Why an Esperanto language computer journal ?

- Because science and Esperanto are international and neutral, which means that they do not belong to any bloc either linguistic or racially, eastern or western, northern or southern, developed or developing, but that both science and Esperanto belong to the whole human race.

If you too are aware of belonging to a worldwide community, and beleive that science must progress for the happiness and prosperity of all peoples, then to bring science closer to everyone, it is best to write in the language which is easiest to learn : Esperanto, a foreign language for everyone, which also does away with all language discrimination by making it necessary for all the inhabitants of the earth to learn at least one foreign language for international communication.

The Editor.

P.S : In your country there is an Esperanto association which will supply further information if desired; its address will be found at the end of this number.

Français :

"Aktuala Komputiko" est une revue internationale spécialisée dans les recherches actuelles dans les différentes disciplines en relation avec les ordinateurs et leur usage.

Dans "Aktuala Komputiko" ne paraîssent que des articles originaux rédigés par des spécialistes, et toujours avec au moins un résumé dans une langue nationale.

Pourquoi une revue spécialisée en Espéranto ?

- Parce que la science et l'Espéranto sont internationaux et neutres, cela veut dire qu'ils n'appartiennent à aucun bloc que ce soit linguistique ou racial, de l'Ouest ou de l'Est, du Nord ou du Sud, des pays évolués ou des pays en voie de développement, mais que la science comme l'Espéranto appartiennent à l'humanité toute entière.

Si vous aussi, vous avez conscience d'appartenir à une communauté universelle et pensez que la science doit progresser pour le bonheur et la prospérité de tous les peuples, il est préférable de publier dans la langue la plus facile à apprendre : l'Espéranto, langue étrangère pour tous les hommes, elle élimime toute discrimination linguistique en obligeant tous les habitants de la terre à apprendre une langue étrangère pour la communication internationale.

Le Rédacteur.

P.S : Dans votre pays il existe une association pour l'Espéranto qui est prête à vous informer si vous le désirez, son adresse se trouve à la fin de ce numéro.

Jen finfine la unua numero de "AKTUALA KOMPUTIKO", ĝi aperas kun preskaŭ unujara malfruiĝo. Tion mi ŝuldas al la manko de referaĵoj por publikigo.

Mi ripetas ke mi ne riskas monon en tiu faka revuo. Mi ne deziras havi abonantaron, ĉar ties administrado estas tro temporaba. Mi havas eblecon por rapide kaj malmulte koste eldoni malgrandan nombron da ekzempleroj, sekve tio tute taŭgas por faka revuo en Esperanto, ĉar ne multaj personoj aĉetus ĝin sed plej grave ekde nun la fakuloj povas senhezite verki en Esperanto, iliaj artikoloj, se ili kontentigas bazajn postulojn aperos en "Aktuala Komputiko". Tiel mi esperas esti rompinta la diablan cirklon : manko de abonantaro implicas mankon de revuo kiu implicas mankon de verkado de fakuloj kiu implicas malprosperon de (faka) lingvo.

Se via lingvo estas grava scienca lingvo ankoraŭ ne uzata komence de tiu numero por prezenti "AKTUALA KOMPUTIKO", mi dankos vin se vi sendos al mi prespretan tradukon de tiu prezento (precipe se via lingvo ne uzas la latinan alfabeton).

Sed pli grave, mi daŭre atendas novajn artikolojn por aperigo en la sekvantaj numeroj de "AKTUALA KOMPUTIKO". La du ĉefaj postuloj estas ke ili estu originalaj kaj ke ili estu nereklamaj tio estas ke ili prezentu neindustriajn aparatojn sed nur aktualajn esplorojn (eble pri estontaj industriaj aparatoj).

Artikoloj devas prezentiĝi tiel :
- supre maldekstre : fako.
- centre supre : titolo.
- sube centre : aŭtornomo.
- sube centre : aŭtoradreso (eventuale).
- sube maldekstre : ŝlosilaj vortoj.
- sube maldekstre : vorto resumo kaj poste resumo en Esperanto.
- sube : resumo(j) en nacia lingvo (almenaŭ unu).
- ekde la posta paĝo : artikolo. 5- ĝis 15-paĝa, nete tajpita, formato A4 kun 2,5 cm-a marĝeno ĉirkaŭe. La paĝoj devas esti numeritaj (prefereble sube).
- fine : bibliografio (eventuale).
- fine : glosaro (eventuale).
- fine : biografio kun nigra-blanka foto.

Artikolon oni devas sendi al la redaktoro.

Mi pretas akcepti senpagajn reklamojn rilatajn al komputiko sed nur sur unu paĝo kaj por unu numero. Kontraŭ pago, post interkonsento, aliaj reklamoj povos aperi en "AKTUALA KOMPUTIKO".

Fako : sonbildaj serviloj

LA PRELEGSERVILO "TISANE"

Christian BERTIN
CCETT AIS
B.P : 59
rue du Clos Courtel
F-35510 CESSON-SEVIGNE
FRANCIO

ŝlosilaj vortoj : servilo, reto, sono, bildo, diĝita transsendo.

Résumé : Cet article présente le serveur de programmes TISANE où un programme est constitué d'un enchaînement d'images fixes et de son. Ce serveur dispose des équipements nécessaires à la saisie, au stockage et à la restitution de son et d'images numériques.

Certains développements en cours sont exposés : connexion au serveur à travers RTC64 (Réseau Téléphonique Commuté à 64 kb/s) et le raccordement d'un DON (Disque Optique Numérique).

Resumo : Tiu artikolo prezentas la prelegservilon TISANE en kiu prelego konsistas el sinsekvo de senmovaj nigra-blankaj bildoj kaj de sono. Tiu servilo disponas pri la aparatoj necesaj por la akiro, la konservo kaj la malakiro de diĝitaj sono kaj bildoj.

Pluraj evoluigaj laboroj estas prezentataj : konekto al la servilo tra la KTR64 (Komutita Telefona Reto je 64 kb/s) kaj la konekto de DOD (Diĝita Optika Disko).

LA PRELEGSERVILO "TISANE"

Kun la evoluo de la mikrokomputiloj kaj de ties konektitaj aparatoj (ekrano anstataŭ papero, diskoj aŭ kasedoj anstataŭ trubendoj aŭ trukartoj), la aparato de la uzanto fariĝas pli kaj pli kompleta kaj pli kaj pli "inteligenta".

Tiel, antaŭ iom da tempo ĉiuj estis kontentaj kiam eblis videbligi signojn sur ekrano, sed nun la ekranoj estas pli kaj pli bonaj kaj sekve la uzanto kutimiĝas utiligi ĝiajn kapablojn kaj postulas pli grandan difinon de la ekranoj (tio estas pli da punktoj en ĉiu linio kaj pli da linioj sur la ekrano) por videbligi pli bonajn kaj pli kompleksajn desegnaĵojn. Fabrikistoj proponas pli kaj pli da koloroj videblaj sur siaj ekranoj kaj la uzanto fariĝas pli kaj pli postulema.

Sed, ne sufiĉas havi grandkapablan ekranon necesas ankaŭ prepari la informojn kiujn oni povas videbligi sur ĝi, jen la tikla problemo, ĉar pretigi belan multkoloran desegnaĵon kun teksto estas temporaba. Pro tio eĉ se la kapabloj de ekranoj rapide kreskas fore postrestas la verkado de interesaj aplikoj.

Aldone al tio, multaj fabrikistoj proponas ankaŭ sonsintezilon aŭ voĉsintezilon (Texas Instruments) sed same interesaj aplikoj kiuj utiligas tiujn kapablojn estas ankoraŭ tro malmultaj pro la tempo necesa por verki ilin.

Pro tio ni interesiĝis pri tio kaj alvenis al elpenso kaj realigo de servilo kiu proponas prelegojn, tio estas sinsekvo de senmovaj nigra-blankaj bildoj kun parolaj komentoj.

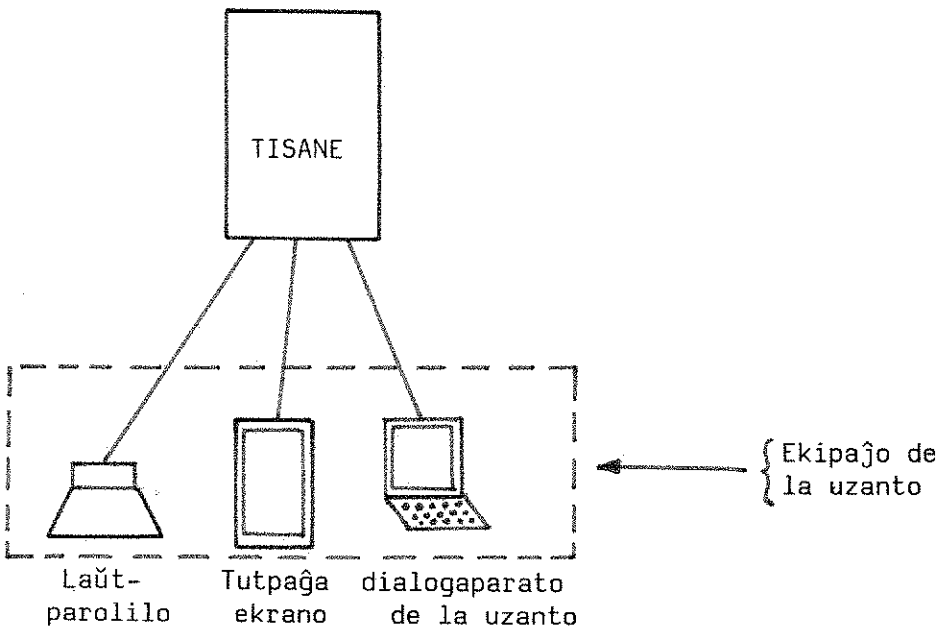
Kiu estas la avantaĝo de tia prelegservilo ? La ĉefa avantaĝo estas la rapido de la produktado de la prelegoj, ĉar ni utiligas telekopiilon por akiri la bildojn kaj analog-al-diĝitan konvertilon por akiri la parolajn komentojn, poste necesas nur registri la sinsekvon de la bildoj kaj komentoj en programon.

Kiam la programo (sinsekvo de bildoj kaj parolaj komentoj) estas preta, ni aldonas ĝin al la listo de la disponeblaj programoj proponataj al la uzantoj, la kreanto tiel ĵus finis sian laboron.

1.0 LA PRELEGSERVILO "TISANE" POR LA DIVERSAJ UZANTOJ.

La nomo "TISANE" signifas en la franca lingvo : Télécopie, Image et Son pour Audiographie Numérique Evolutive, kio signifas en Esperanto : Telekopio, Bildo kaj Sono por Aŭdgrafikio Diĝita Evoluebla.

TISANE estas servilo de programoj, programo estas sinsekvo de senmovaj nigra-blankaj bildoj kun parolaj komentoj. Ekzemplo de tiatipa programo estas prelego.



Bildo 1. La prelegservilo vidata de la uzanto.

La ekipaĵo de la uzanto kiu estas nun senpere konektita al la servilo konsistas el :

° dialogaparato (por komandi la servilon kaj por ricevi la servmesaĝojn).

° tutpaĝa ekrano (por videbligi la senmovajn nigra-blankajn bildojn).

° laŭtparolilo (por la eliro de la parolaj komentoj).

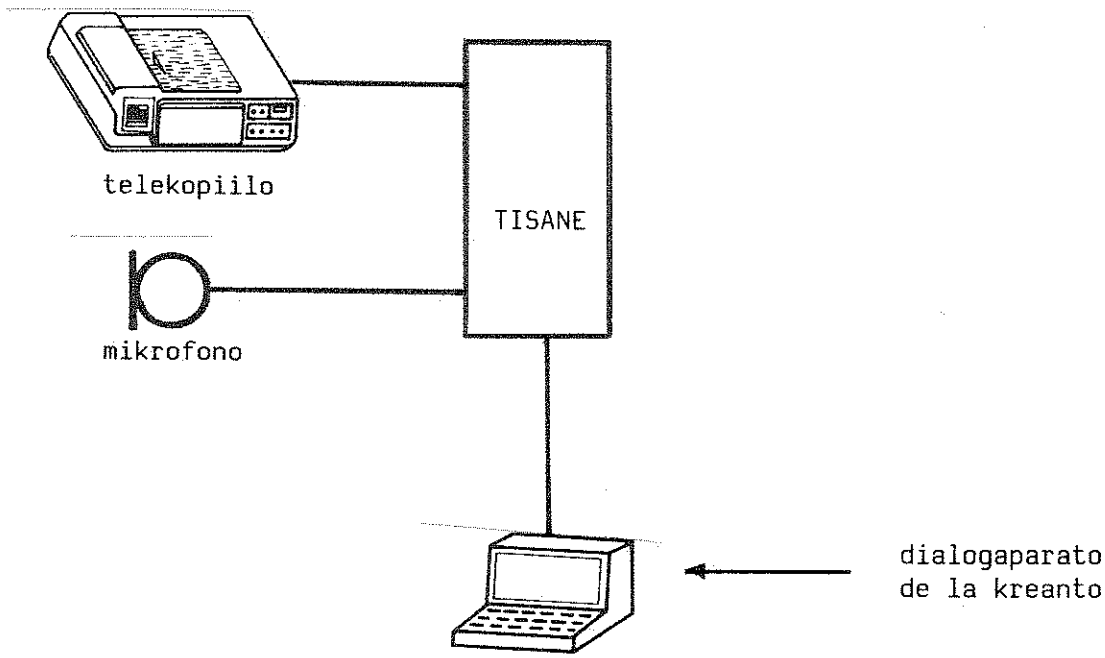
La bildo 1 prezentas la prelegservilon vidatan de la uzanto. La bildo 2 prezentas la prelegservilon vidatan de la programkreanto.

Ankaŭ la iloj necesaj por la kreado de programoj estas senpere konektitaj al la prelegservilo TISANE, ili konsistas el :

° telekopiilo TEGEFAX 2000 el la firmao THOMSON-CSF (por la akiro de la bildoj).

° analog-al-diĝita konvertilo realigita en nia oficejo (por la akiro de la parolaj komentoj per mikrofono kaj amplifilo).

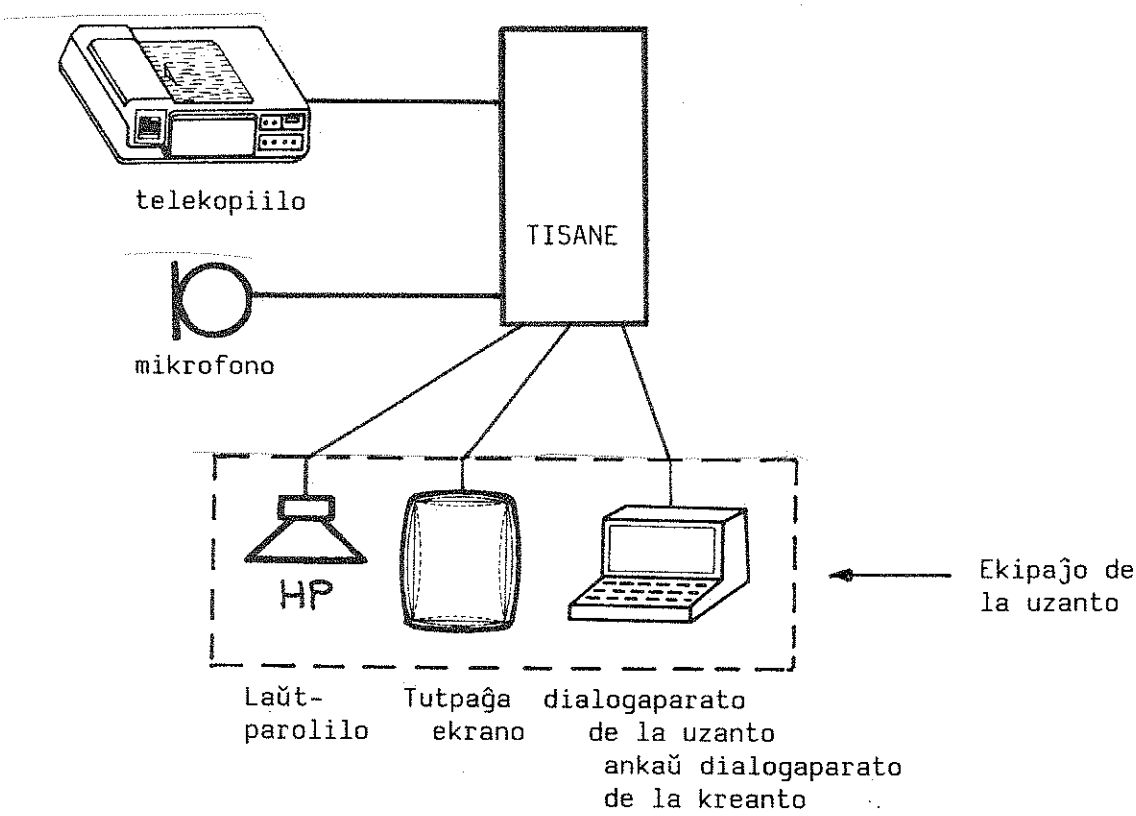
° dialogaparato uzata de la kreanto (por krei la sinsekvon de la bildoj kaj komentoj akiritaj per la du aliaj iloj).



Bildo 2. La prelegservilo vidata de la kreanto.

2.0 PREZENTO DE LA EKZISTANTA PRELEGSERVILO "TISANE".

La nun ekzistanta prelegservilo "TISANE" estas videbla sur la bildo 3. Tiu bildo, kvankam kompleta estas malpli klara ĉar ĝi kunigas ĉiujn aparatojn necesajn por nia eksperimento en la sama maŝino. Fakte ni devus havi du maŝinojn, unu por la produktado de programoj kaj la alia por oferti ilin al la uzantoj kiel montras bildoj 1 kaj 2.



Bildo 3. Konfiguro de la prelegservilo "TISANE".

Ni nun priskribos la diversajn interfacojn ekzistantajn inter la eksteraj aparatoj kaj la servilo.

2.1 LA INTERFACO KUN LA TELEKOPIILO.

Nia laboratorio elpensis kaj realigis interfacan tabulon por la servilo TISANE por konekti la telekopiilon TEGEFAX 2000 de la firmao THOMSON-CSF per ties konektilo V24.

La telekopiilo TEGEFAX 2000 skenas la surfacon de paĝo je formato A4 po 1728 punktoj en ĉiu linio de paĝo kaj po 1144 linioj, tio estas entute preskaŭ 2 milionoj da punktoj por ĉiu paĝo je formato A4.

Tiuj 2 milionoj da punktoj estas transsendataj tra la konektilo V24 kiel samnombraj bitoj al la servilo.

Pro la fakto ke la ekrano konektita al la servilo en la ekipaĵo de la uzanto ne havas la saman difinon kiel la telekopiilo, ne necesas konservi pli da punktoj ol estas sur la ekrano, sekve la servilo TISANE aplikas reduktalgoritmon por malpli grandigi la nombron de la registritaj bitoj je la nombro de la punktoj videblaj sur la ekrano.

Ne sufiĉas ke la difino de du aparatoj estu la sama (tio estas sama nombro de punktoj en linio kaj sama nombro de linioj) necesas ankaŭ ke la vidsurfacoj estu samproporciaj, se ne la bildoj kiujn vi akiras per unu aparato aperos aliformaj sur alia aparato. Kiel ekzemplo, cirklo sur la unua aperos kiel ovalo sur la dua, kvadrato sur la unua aperos kiel ortogramo; tiu problemo povas esti grava por mapoj aŭ portretoj. Tio estis unu el laj problemoj kiujn ni devis solvi.

2.2 LA INTERFACO KUN LA SONAMPLIFILO.

La analoga sono estas akirata tra mikrofona, tunera aŭ magnetofona enirejo.

La sono estas specimenata je 8-kHz-a frekvenco kaj la akirita specimeno okupas 8 bitojn, pro tio la bitflua rapido estas 64 kb/s kiujn ni devas registri sur magnetan diskon. Pro la specimenad-frekvenco ni povas akiri nur la sonojn kiuj troviĝas en la telefona bendo, tio estas proksimume 0-4 khz.

Nia disko ne havas tre grandan kapaciton sed ĝi sufiĉas por nia eksperimento. Post sukcesa eksperimento, ni intencas utiligi diĝitan optikan diskon (DOD).

2.3 LA INTERFACO AL LA TUTPAĜA EKRANO.

Tiun ekranon ni aĉetis de la firmao MOTOROLA, ĝia difino estas la sekvanta : 800 punktoj en ĉiu linio kaj 1024 linioj. Kiel ni diris por la interfaco al la telekopiilo, la vidsurfaco de tiu ekrano ne estas samproporcia al tiu de la telekopiilo.

La regtabulo kiu troviĝas en la servilo por tiu ekrano estas fabrikita de la firmao NEC kaj havas kolorajn kapablojn, ĝi enhavas

memoron por tri koloraj planoj (ruĝa, verda kaj blua), sed nia ekrano ne estas kolora.

2.4 LA INTERFACO AL LA LAŬTPAROLILO.

Por tio ni uzis la saman regtabulon realigitan en nia oficejo kiu plenumas la malan funkcion de la akiro de sonoj, tio estas diĝit-al-analoga konvertilo. Tiuj du funkcioj ekzistas sur la sama tabulo por nia eksperimento, sed evidente tiu tabulo estas necesa en la maŝino por produkti la programojn, ĉar oni devas reaŭskulti la programojn antaŭ ol liveri ilin al la servilo. Nur la funkcio sonsintezado estas necesa en la ekipaĵo de la uzanto.

2.5 LA INTERFACO AL LA DIALOGAPARATO.

Tio estas tutsimpla interfaco kiel oni vidas en ĉiuj komputiloj, la aparato estas ekranklavaro kiu videbligas 25 liniojn el 80 signoj.

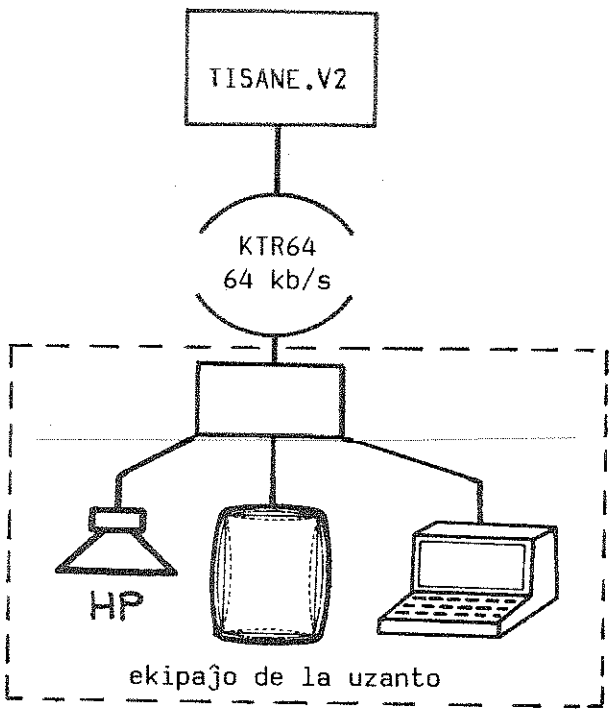
3.0 LA EVOLUO DE LA PRELEGSERVILO "TISANE".

Pluraj eblecoj ekzistas antaŭ ni por plua laboro sur tiu kampo. Unue necesas rimarki ke sufiĉas ŝanĝi kaj la bildakirilon (telekopiilo) kaj la ekranon de la uzanto por prezenti samspecajn kolorajn programojn, la funkcioj restas la samaj. Due ni devas konsideri la tuj ekzistontajn retojn, en Francio tio estas la komutita telefona reto je 64 kb/s (KTR64), en tiu kunteksto, nia servilo estas bona kandidato por tiu reto, ĉar ni bezonas rapidan bitfluon kaj tiu reto ofertas ĝin al ni.

3.1 LA KONEKTO AL LA KOMUTITA TELEFONA RETO JE 64 KB/S (KTR64).

Por tio necesas aldoni al la servilo funkcion por ebligi interfacon al la komutita telefona reto je 64 kb/s.

Necesas ankaŭ apartigi la ekipaĵon de la uzanto disde la servilo, tio signifas ke ni devas realigi veran kompletan aparaton por la uzanto. Tiu aparato konsistos el mikrokomputilo kun komunika funkcio por rilati al la KTR64 kaj kun regfunkcio por regi la eligajn ilojn : laŭtparolilo, tutpaĝa ekrano kaj dialoga ekranklavaro. La bildo 4 montras la novan konfiguron de uzado de la prelegservilo TISANE.



Bildo 4. Prelegservilo por KTR64.

Post la fabriko de nova ekipaĵo de uzanto kaj programado de la komunikprogramo inter la servilo kaj la uzanto, necesas ankaŭ labori pri la bitkvanto necesa por bona komunikado. La interligo inter la uzanto kaj la servilo funkcias je 64 kb/s, tio signifas ke se oni deziras sendi kaj bildon kaj sonon, neniu el la du fluoj devas atingi 64 kb/s ĉar necesas pliaj bitoj por indiki nun estas sono aŭ nun estas bildo.

Nia sono necesigas 64 kb/s sekve ĝi ne estas rekte uzebla por la KTR64 escepte se oni akceptas ke la interligo inter servilo kaj uzanto konsistu el du lineoj : unu por la sono kaj la alia por la bildo kaj la dialogoj kun la uzanto. Tion ni ne pretas akcepti, sekve ni devas trovi alian solvon : redukto de la bitkvanto.

3.1.1 LA REDUKTO DE BITKVANTO POR LA SONO.

Post laboratoriaj studoj en kiu oni provis redukti la bitkvanton de ĉiu specimeno (ni memorigu ke la specimeno estas 8-bita kaj la specimenad-frekvenco estas 8 kHz). Ni simple ne konsideris la malplej gravajn bitojn, komence unu, poste du kaj tiel plu. Praktike tio signifas ke la sono normale akirita kaj registrita je 64 kb/s en la magneta disko de TISANE estis modifita antaŭ la sendo al la sonsintezilo. Tiu modifo konsistis el nuligo de la malplej grava(j) bito(j) de ĉiu specimeno.

validaj bitoj (X)	bitkvanto	kvalito
XXXXXXXX	64 kb/s	bona
XXXXXXX0	56 kb/s	bona
XXXXXX00	48 kb/s	bona
XXXXX000	40 kb/s	ekaŭdo de bruo

La provoj montris ke fonbruo eksentiĝas kiam la 3 malplej gravaj bitoj estas nuligitaj, sekve sen problemo kun nia ekzistanta sonakirilo, ni povas malpli grandigi la bitkvanton je 48 kb/s sen perdo sentebla en la sonkvalito.

Post tio oni povas ankoraŭ redukti la bitkvanton necesan por la sono per detekto de la silentoj kaj redukto de la bitoj por kodi ilin. Por tio ni ankaŭ faris eksperimenton en kiu ni konsideris la sonon laŭ blokoj de specimenoj kaj ni fiksis sojlon por detekti ĉu specimenbloko estas silentbloko aŭ sonbloko. Sojlo estas fiksita kiel nombro de specimenoj kiuj troviĝas ekster segmento de valoroj.

Ni faris kvalitan mezuradon kun diversaj bloklongoj kaj kun diversaj valoroj de la sojlo.

Ni rimarkis ke ne sufiĉas fari tion por bone redukti la bitkvanton sen perdo de sonkvalito, necesas ankaŭ ne abrupte salti de silento al sono, pro tio ni decidis ne redukti blokon detektitan kiel silentbloko en du kazoj :

° silentbloko kiu postas sonblokon ne devas esti reduktita.

° silentbloko kiu antaŭas sonblokon ne devas esti reduktita.

Kun tiuj rimarkoj, ni faris la sekvantajn provojn :

bloklongo	sojlo	redukto	kvalito
128	7	28%	nebona
255	7	25%	nebona
512	8	21%	nebona
512	16	21%	nebona
1024	16	16%	bona
1024	32	18%	bona
1024	48	19%	nebona
1024	64	21%	nebona

3.1.2 LA PROTOKOLO INTER LA SERVILO KAJ LA UZANTO.

Inter la servilo kaj la uzanto estos tri bitfluoj : unu por la laŭtparolilo (sono), alia por la tutpaĝa ekrano kaj la lasta por la dialogo inter la uzanto kaj la servilo.

Ni devas sendi tiujn tri bitfluojn per la sama interligo kiu konsistas el nur unu lineo, konsekvence ni devas multipleksi la tri bitfluojn.

Por la interŝanĝo de informoj inter aparatoj jam ekzistas protokoloj difinitaj kaj normitaj de la CCITT aŭ de ISO por la interkonekto de malfermitaj sistemoj (IMS). Sed la normado de tiuj protokoloj ankoraŭ ne atingis la nivelon kiu interesas nin, ili ĝis nun ne konsideras la kunekziston de diversaj informfluoj por la sama apliko. Sekve ni devos labori en tiu kadro por la enkonduko de la necesaj rimedoj por identigi

la diversajn fluojn kaj certigi ilian sinkronigon ĉe la uzanto se la apliko postulas tion.

Tio havos efekton pri la starigo de komuniko inter la servilo kaj la uzanto ĉar dum tiu fazo, la du aparatoj interŝanĝas informojn pri siaj kapabloj (ĉu kolora aŭ ne, ĉu alfafotografa aŭ ne, ĉu alfageometria aŭ ne, ĉu sona aŭ ne kaj laŭ kiu kodo kiam ekzistas pluraj kodoj por la sama informtipo).

3.2 KONEKTO DE DIĜITA OPTIKA DISKO (DOD) AL TISANE.

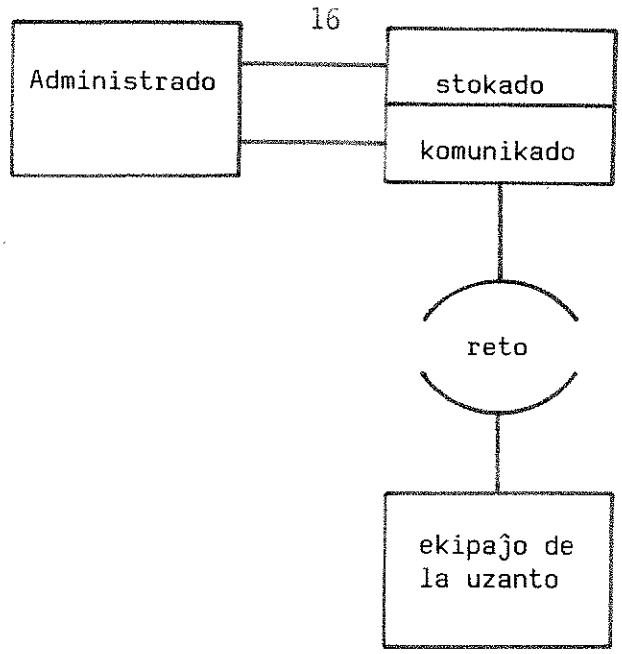
La kvanto de informo kiun oni devas registri por sono kaj bildo estas tiel granda eĉ post bona reduktalgoritmo ke la solvo per diĝita optika disko estas interesa. La kapacito de DOD estas minimume 1 Go (unu Gigabitoko). Jam ekzistas pluraj fabrikistoj por tiuj medioj (Hitachi, Shuggart, Thomson, Philips...) Por la administrado de DOD ni uzas akompanan magnetan diskon, tio estas kutima disko kiu akompanas la DOD dum ĝia plenigo, ĉar ĝi entenas la katalogon de ĉio kio estas registrita sur la DOD. Tiu solvo faciligas la administradon de DOD, ĉar necesas memori ke la DOD estas nereregistrebla medio : tio signifas ke se iu sektoro de la disko estas registrita, oni ne plu povas modifi ĝian enhavon, tamen oni povas senvalidigi ĝin. Sekve se la katalogo estus registrita samtempe kiel la rikordaroj sur la DOD ĉiu modifo (aldono, forigo) necesigus la registradon de nova katalogo en alia parto de la disko kaj plie oni devus havi manieron por retrovi la katalogon post restarto de la servilo. Do, ĉiu modifo de la enhavo de la DOD estos registrata sur la akompana disko.

Kiam la DOD estas preskaŭ plena, tiam ties katalogo estas ankaŭ registrata sur la DOD por igi la DOD aŭtonoma. Tiu aŭtonoma DOD povas esti multekzemplere kopiata por livero al la diversaj serviloj TISANE.

3.3 PARTIGO DE LA SERVILO TISANE EN DU MASINOJN.

La servilo TISANE fakte konsistas el tri gravaj funkcioj kiuj estas :

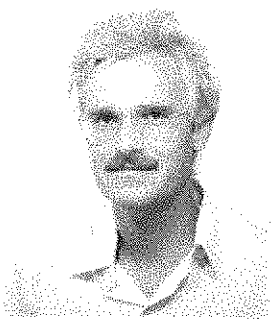
- funkcio "stokado" kiu prizorgas la registradon kaj la legadon de la informoj kiaj sonoj kaj bildoj.
- funkcio "administrado" kiu prizorgas la dialogon kun la uzanto kaj la administradon de la datumbanko.
- funkcio "komunikado" kiu prizorgas la interŝanĝon de informoj inter la servilo kaj la uzanto.



Bildo 5. Partigo de la servilo en du maŝinojn.

La bildo 5 klare montras la partigon de la prelegservilo TISANE en du maŝinojn kvankam ni identigis tri funkciojn. Tio estas pro la fakto ke la funkcioj "stokado" kaj "komunikado" postulas grandan bitfluojn inter si, tial ni decidis kunigi tiujn du funkciojn en la sama maŝino.

La avantaĝo de tia strukturo estas, post difino de taŭga komunikprotokolo inter la du maŝinoj, la sendependigo de la du maŝinoj kiuj postulas malsamajn kvalitojn kaj kies teknologio ne samrapide evoluos. Tio estas bona ankaŭ pro la fakto ke fabrikistoj povos esti sendependaj, kaj tio estas realo en la nuna merkato : iuj fabrikistoj estas kompetentaj en "datumbanka administrado" dum aliaj estas pli kompetentaj en "stokado" aŭ en "komunikado".



BIOGRAFIO : Christian Bertin naskiĝis en 1949, akiris inĝenieran diplomon pri informatiko en 1972 kaj doktoran inĝenieran diplomon en 1984 pri datumtranssenda reto per satelito en Indonezio. De 1976, li laboras en CCETT (Centre Commun d'Etudes de Télédiffusion et de Télécommunications) kiu estas Esplorcentro komuna pri televido kaj telekomunikoj. Li estras laboratorion kiu planas la servilojn por la flurapidajn retojn ekde 64 kb/s.

programig-teknikoj

LA PROGRAMIGO DE STORO-KOPIOJ

Boris Gärtner
Schöttlstraße 12
D-8000 München 70

ŝlosilaj vortoj:

storo-kopio, rekursivaj proceduroj, procedur-valoraj parametroj, procedur-listoj, PASCAL, ALGOL 68, LISP, ALGOL 60, MODULA-2, ADA

Resumo:

Estas klarigataj du teknikoj de la programigo de storo-kopioj kaj estas klarigataj iuj limigoj de alt-nivelaj programiglingvoj.

Two methods for the generation of dumps are discussed and some restrictions of high-level algorithmic languages are pointed out.

Pri analizo de (eble eraraj) programoj ofte estas helpema informo, kiu respegulas la enhavon de la tuta storo. Post eraro la storo-administraj sistemoj de multaj programig-sistemoj eligas storo-kopion. Tiu artikolo traktas la programigo de storo-kopioj en alt-nivelaj algoritmaj lingvoj.

Kompleta storo-kopio konsistas el informo pri ĉiuj procedur-enkarniĝoj, kiuj ekzistas en la momento de la eligo de la storo-kopio. Tial ĉiu proceduro devas posedi proceduron, kiu unue eligas informon pri la objektoj de la proceduro mem, kaj due efikas eligon de informo pri ĉiuj antaŭealvokitaj proceduroj. La unua ekzemplo klarigas tion.

La aldonita procedurvalora parametro efikas la kunligon de la kreitaj procedur-enkarniĝoj. Post la eligo de la informo pri iu procedur-enkarniĝo la alvoko de la procedurvalora parametro efikas la daŭrigon de la eligo. La elig-proceduroj fakte formas duplikaton de la dinamika referenc-ĉeno (angle: dynamic reference link).

La dua ekzemplo ilustras iomete plimalfacilan situacion: Procedur-valora parametro de iu proceduro devas ricevi aldonan procedur-valoran parametron, kiu permesas longigon de la duplikato de la dinamika referenc-ĉeno. Rigardu, ke la aldono de procedur-valora parametro al proceduro, kiu estas uzata kiel para-

metro de alia proceduro ne estas permesata en la lingvo, kiu estis kreata de profesoro Wirth kaj kiu estis nomata de li "PASCAL". Tiu ĉi limigo estis forigata en la internacie normita lingvo ISO-PASCAL.

Grava malavantaĵo de la klarigita tekniko estas, ke por ĉiu ajn procedur-enkarniĝo dum la eligado de la storo-kopio necesas kreado de nova procedur-enkarniĝo. Tial la klarigita tekniko estas storo-konsumema. La esprimiloj de la lingvo ALGOL 68 tamen permesas senrekursivan ekvivalenton de tiu tekniko. Nun la inform-eligaj proceduroj estas administraj kiel listo, kiu estas inspektata per la proceduro "eligu kopio". Pri tiu senrekursiva varianto la rekursiva tip-difino

```
mode kopiillisto = struct (proc void kopio,
                           ref kopiillisto antaŭulo);
```

posedas esencan signifon. Egale esencan signifon posedas la kreilo loc , kiu ofertas eblecon krei objekton en la privata storo-segmento de proceduro. Tio gravas, ĉar kreado de procedur-posedantaj objektoj en komuna stor-parto (angle: heap) maleblas, ĉar tiuj objektoj ne ĉesas ekzisti post la malkreado de la stor-segmentoj, kiuj ili nomigas kiel iliajn ĉirkaŭaĵojn.

La tria ekzemplo ilustras, kiel estas konstruata, uzata kaj malkonstruata listo de proceduroj.

La plejmulto de la algoritmaj lingvoj ne posedas esprimilojn por la konstruado de listoj, kies elementoj estas proceduroj. En la lingvo MODULA-2 ero de datum-strukturo povas esti proceduro, sed valoro de tiu strukturero devas esti proceduro, kiu estas difinata en la difin-parto de la programo mem. Tio estas kompreneble, se oni atentas, ke - kontraŭe al ALGOL 68 - la lingvo MODULA-2 ne posedas objekt-kreilon, kiu kapablas krei objekton en privata stor-segmento de proceduro. Krome ankaŭ proceduroj, kiuj estas uzataj kiel parametroj, devas esti difinata en la difin-parto de skribanta en MODULA-2 programo. Tial MODULA-2 ne permesas nek rekursivan nek senrekursivan storo-kopiilon.

La lingvo LISP permesas listoj kun iutipaj objektoj. (Tio estas

tute kompreneble, se oni rememoras, ke LISP posedas nur unu strukturon, kiu representas programojn kaj datumojn.) Storo-kopiiloj povas esti programataj, se la lingvo posedas ilon, kiu permesas kunligi LISP-funkcion al difina ĉirkaùaĵo, t.e., kiu permesas cirkumigi la regulojn pri la valor-retrovo en la lingvo LISP. Tiu ligilo estas la funkcio FUNCTION (kiu en kelkaj LISP-sistemoj nomigas CLOSURE). La alvoko (FUNCTION <esprimo>) resultas ĝian argumenton ligita al ĉirkaùaĵo, en kiu estis ellaborata la alvoko.

LISP ebligas la programigon de ambaù rekursivaj kaj senrekursivaj storo-kopiiloj. (Ekzemploj 4 kaj 5) Por uzo kun senrekursiva storo-kopiilo ĉiu LISP-funkcio estas transformata laù la regulo:

```
      (LAMBDA <parametro-listo9  <esprimo>)
igas
      (LAMBDA <parametro-listo>
       ((LAMBDA (KOPIILLISTO)  <esprimo> )
        (CONS (FUNCTION (LAMBDA ()  <elig-esprimoj>)
              KOPIILLISTO
      )))
```

Sur bazo de tiu regulo eblas programigi funkcion, kiu transformas LISP-esprimon per aldono de informilo. Simpla transformilo estas aldona kiel sesa ekzemplo.

Estas rimarkinde, ke tiel malriĉe strukturata lingvo kiel LISP permesas konstruojn, kiuj maleblas en aliaj lingvoj. Fakte, oni devas konstati, ke la plejmulto de algoritmaj lingvoj ne ofertas sufiĉan riĉecon da proceduraj tipoj por la programigo de storo-kopiiloj. Ni ilustras tion kun rimarkoj pri la lingvoj ALGOL 60 kaj ADA. Ambaù lingvoj ne permesas programigi storo-kopiiloj.

Pri ALGOL 60 tio estas kaùzata per la mekanismo de la "nom-transdono" (angle: call by name). Fakte nom-transdona parametro estas paro de senparametraj proceduroj, kiuj ne povas posedi privatajn objektojn (vidu ekzemplo 7).

Pri ADA la koncepto de generaj proceduroj ne estas sufiĉe ĝener-

ala. Estis imitata malnova manko de la lingvo PASCAL: Proceduro, kiu estas uzata kiel parametro, ne povas posedi procedur-valorajn parametrojn. Tial la aro de haveblaj proceduraj tipoj estas tiel limigata, ke duplikato de la dinamika referenc-ĉeno ne povas esti longigata el proceduro al la alvokoj de ĝiaj procedur-valoraj parametroj. Anstataŭe esprimilo nomata "erar-servilo" (angle: exception handler) ebligas programigon de malkonstruantaj storo-kopiiloj. Aktivig-vico de erar-serviloj ĉiam sekvas al la dinamika referenc-ĉeno (Ekzemplo 8).

Malkonstruantaj storo-kopiiloj programigeblas ankaŭ an aliaj lingvoj per simpla uzo de salt-instrukcioj. Ekzemplo 9 klarigas tion. ĉu tio ĉi stilo plaĉas al ĉiuj ?

LITERATURO

Backus, J.W. kaj aliaj:
Revised Report on the Algorithmic Language ALGOL 60
Numerische Mathematik 4, 420 - 453 (1963)

esperantlingva traduko:

Revisiita raporto pri la algoritma lingvo Algol 60
1983 Budapesto, Hungara Esperanto-Asocio

McCarthy, John:
Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I
Communications of the ACM 3, 185 - 195 (1960)

van Wijngaarden, Aadrian; kaj aliaj:
Revised Report on the Algorithmic Language ALGOL 68
1976 Springer Verlag Berlin Heidelberg New York

Jensen, Kathleen; Wirth, Niklaus:
Pascal User Manual and Report
1975 Springer Verlag Berlin Heidelberg New York

esperantlingva libro pri la lingvo PASCAL:

Bertin, Cristian:
La programlingvo Pascal
1982 Rennes

Wirth, Niklaus:
Programming in Modula-2
1982 Springer Verlag Berlin Heidelberg New York

--
The Programming Language ADA
Reference Manual
U.S. Department of Defense (1980, revisiita 1982)

E K Z E M P L O J

Programo sen stoko-kopiilo:

```
program ekzemplo1 (output);
  var  valoro : real;

  function f (a : integer) : real;

  begin
    if a > 0 then f := a * f (a - 1)
             else f := 1

  end;
begin
  valoro := f (10);
  writeln (valoro)
end.
```

Sama programo kun stoko-kopiilo:

```
program ekzemplo1 (output);
  var  valoro : real;
  procedure kopio1;
  begin
    writeln (' la programo mem.');   writeln;
    writeln ('Valoro de ''valoro : real'' estas ',
             valoro);
    writeln;    writeln (' *** fino ***')
  end;
  function f (a : integer; procedure kopio) : real;
    procedure kopiof;
    begin
      writeln (' la funkcio f.');  writeln;
      writeln ('Valoro de ''a : integer'' estas ', a);
      write   ('Antaùealvokita proceduro estas');
      kopio
    end;
  begin
    if a > 0 then f := a * f (a - 1, kopiof)
             else begin
                    kopiof;     f := 1
                  end
  end;
begin
  valoro := f (10, kopio1);
  writeln (valoro)
end.
```

(Ekzemplo 1: simpla rekursiva proceduro)

```
function int (a, b, h : real;
              function f (x : real)
                       : real)
           : real;
var  valoro : real;

begin
  (*  ....  *)
  valoro := f (a + h);
  (*  ....  *)
end;
```

```
function int (a, b, h : real;
              function f (x : real; procedure kopio)
                       : real;
              procedure kopio) : real;
  var   valoro : real;
  procedure kopioint;
  begin
    (*  eligo  de informo pri a, b, h, valoro   *)
    kopio   (*  daùrigo de inform-elmetado      *)
  end;
begin
  (*   ....  *)
  valoro := f (a + h, kopioint)
  (*   ....  *)
end;
```

(Ekzemplo 2: proceduro kun procedur-valora parametro)

```
begin
  mode kopiilisto = struct (proc void kopio,  ref kopiilisto  antaùulo);

  proc (ref kopiilisto) void  eligu kopio =
       (ref kopiilisto  komenco)  void :
  begin
    ref ref kopiilisto  nuna elemento = loc ref kopiilisto := komenco;

    while ref kopiilisto (nuna elemento) isnt nil
    do
      kopio of nuna elemento;    co  eligo de la informo pri unu procedur-enkarniĝo  co
      nuna elemento := antaùulo of nuna elemento       co  antaŭiro en la listo      co
    od
  end;

    comment   sekvas proceduro por la numera integraligo kun storo-kopiiloj    comment

  proc (proc (real, ref kopiilisto) real,  real,  ref kopiilisto)  real  integralo =
       (proc (real, ref kopiilisto) real  f,      co  integrato  co
        real  epsilono,                           co  ekzaktec-limo    co
        ref kopiilisto  antaùaj alvokoj ) :
  begin
    co   ripeta numera integraligo kun ĉifoja duonigo de la paŝ-longo  ĝis atingo    co
    co   de difinita ekzakteco. integralig-intervalo estas la intervalo  0 .. 1 .    co

    ref int i = loc int,
            d = loc int;
    ref real n = loc real,
             t = loc real,
             s = loc real;
    ref kopiilisto informilo = loc kopiilisto :=
        (void : begin
                  print (("  la proceduro integralo.", newline));
                  print (("  valoro de la parametro epsilono estas "), epsilono, newline));
                  print (("  valoro de i estas ", i, newline));
                  print (("  valoro de d estas ", d, newline));
                  print (("  valoro de n estas ", n, newline));
                  print (("  valoro de t estas ", t, newline));
                  print (("  valoro de s estas ", s, newline));
                  print (("  la antaùealvokita proceduro estas "))
                end,
         antaùaj alvokoj);

    t := f (0.0, informilo);      t := (t + f (1.0, informilo))/2.0;      d := 1;
    while
      s := 0.0;     i := 1;
      while not (i > d)
      do
        s +:= f ((2*i - 1)/d/2, informilo);      i +:= 1
      od;
      n := (t + s/d)/2;
      if abs (n - t) <= epsilono then false
                                 else t := n;    d +:= d;    true
      fi
    do
      skip
    od;
    n        co    ellaborita resulto    co
  end;
```

(Ekzemplo 3: senrekursiva storo-kopiilo)

```
  co   la informeligilo de la programo   co
ref kopiiiiisto  programinformo = loc kopiiiiisto
                 := (void : begin
                              print ((" la programo mem.", newline))
                            end,
                    nil);
  co   unua alvoko:  integrato estas la funkcio   (1 - x)/(1 + x*x)      co
print
 (integralo ( (real x, ref kopiiiiisto informo)  real :
              begin
                ref kopiiiiisto loka informo = loc kopiiiiisto
                 := (void : begin
                              print ((" sennoma funkcio.", newline));
                              print ((" valoro de parametro x estas ", x, newline));
                              print ((" la antaŭealvokita proceduro estas "))
                            end,
                    informo);
                eligu kopio (loka informo);   co  storo-kopio eligita  co
                (1 - x)/(1 + x*x)
              end,
              1.0E-6    co   ekzakteco   co  ,
              programinformo
 )          );
  co   dua alvoko:   dufoja integralo de   x * y / (1 + x*x)          co
print
 (integralo
   ( (real y, ref kopiiiiisto informo1) real :
     begin
       ref kopiiiiisto loka informo = loc kopiiiiisto
        := (void : begin
                     print ((" unua sennoma funkcio.", newline));
                     print ((" valoro de parametro y estas ", y, newline));
                     print ((" la antaŭealvokita proceduro estas "))
                   end;
            informo1);
       integralo ( (real x, ref kopiiiiisto informo2) real :
                   begin
                     ref kopiiiiisto loka informo = loc kopiiiiisto
                      := (void : begin
                                   print ((" dua sennoma funkcio.", newline));
                                   print ((" valoro de parametro x estas ", x, newline));
                                   print ((" la antaŭealvokita proceduro estas "))
                                 end,
                          informo2);
                     eligu kopio (loka informo);   co  storo-kopio eligita  co
                     (x*y)/(1 + x*x)
                   end,
                   1.0E-6    co   ekzakteco   co  ,
                   loka informo)
     end,
     1.0E-6     co   ekzakteco    co  ,
     programinformo))
end
```

(Ekzemplo 3, daŭrigo kaj fino)

```
(DEFFUN  $LAST
 (LAMBDA (L)
  (COND ((NULL (CDR L)) L)
        (T ($LAST (CDR L)))
)))
```

Ekzemplo de erara uzo:

```
($LAST '(A B . C))
*** eraro en la funkcio CDR
*** argumento estas atomo: C
```

```
(DEFFUN  $LAST
 (LAMBDA (L INF)
  ((LAMBDA (LOKINF)
    (COND ((NULL (ERRORSET '(CDR L) NIL))
           (LOKINF)  % storo-kopio eligita %
           NIL
          )
          ((NULL (CDR L)) L)
          (T ($LAST (CDR L) LOKINF))
    ))
   (FUNCTION (LAMBDA ()  % informeligilo %
             (PROGN (PRINT '(VALORO DE L ESTAS))
                    (PRINT L)
                    (PRINT '(ANTAŬA ALVOKO)) (INF)
))))         ))
```

Ekzemplo de erara uzo:

```
($LAST '(A B . C) '(LAMBDA () (PRINT 'FINO)))
(VALORO DE L ESTAS)
C
(VALORO DE L ESTAS)
(B . C)
(VALORO DE L ESTAS)
(A B . C)
FINO
= NIL
```

(Ekzemplo 4: rekursiva storo-kopiilo en LISP)

```
(SET 'KOPIILLISTO NIL)  % iniciato %

(DEFFUN ALVOKINFORMO
        (LAMBDA (INFORMILLISTO)
         (PROG (NUNA-ELEMENTO)
               (SET 'NUNA-ELEMENTO INFORMILLISTO)
          L    (AND (NULL NUNA-ELEMENTO)
                    (PRINT '(FINO DE LA ALVOKINFORMO))
                    (RETURN NIL)
               )
               ((CAR NUNA-ELEMENTO))   %  eligo de la informo pri unu alvokita funkcio  %
               (SET 'NUNA-ELEMENTO (CDR NUNA-ELEMENTO))
               (GO L)
)       ))

(DEFFUN $LAST (LAMBDA (L)
               ((LAMBDA (KOPIILLISTO)
                 (COND ((NULL (ERRORSET '(CDR L) NIL)) (ALVOKINFORMO KOPIILLISTO) NIL)
                       ((NULL (CDR L)) L)
                       (T ($LAST (CDR LISTO)))
                ))
                (CONS (FUNCTION (LAMBDA ()  %  nova eligilo  %
                                (PROGN (PRINT '(VALORO DE L ESTAS)) (PRINT L))
                      )         )
                      KOPIILLISTO
)             )))
```

(Ekzemplo 5: senrekursiva storo-kopiilo en LISP)

```
(DEFFUN INFORMFUNKCIO
        '(LAMBDA (ARGUMENTLISTO)
          (SUBST (APPEND '(PROGN)
                         (MAPCAR ARGUMENTLISTO
                                 '(LAMBDA (X)
                                   (SUBST X '$ '(PROGN (PRINT '(VALORO DE $ ESTAS))
                                                       (PRINT $)
                 )       )        ))            )
                 '$
                 '(FUNCTION (LAMBDA () $))
)        ))

(DEFFUN ALDONI
        '(LAMBDA (ESPRIMO)
          (LIST (LIST 'LAMBDA '(KOPIILLISTO) (TRANSFORMI (CADDR ESPRIMO)))
                (LIST 'CONS (INFORMFUNKCIO (CADR ESPRIMO)) 'KOPIILLISTO)
)        ))

(DEFFUN TRANSFORMI
        '(LAMBDA (ESPRIMO)
          (COND ((ATOM ESPRIMO) ESPRIMO)          %  Atomo restas sentransformata  %
                ((AND (ATOM (CAR ESPRIMO))        %  kvotata esprimo ankaù restas  %
                      (EQ (CAR ESPRIMO) 'QUOTE)   %  sentransformata               %
                 )
                 ESPRIMO)
                ((AND (ATOM (CAR ESPRIMO))        %  FQUOTE anstataùas QUOTE antaù %
                      (EQ (CAR ESPRIMO) 'FQUOTE)  %  LAMBDA-esprimo                 %
                 )
                 (COND ((LITATOM (CADR ESPRIMO)) (CONS 'QUOTE (CDR ESPRIMO)))
                       (T (LIST 'QUOTE (TRANSFORMI (CADR ESPRIMO))))
                ))
                ((AND (ATOM (CAR ESPRIMO)) (EQ (CAR ESPRIMO) 'LAMBDA))
                 (LIST 'LAMBDA (CADR ESPRIMO) (ALDONI ESPRIMO))
                )
                (T (MAPCAR ESPRIMO 'TRANSFORMI))
)        ))
```

(Ekzemplo 6: simpla transformilo por LAMBDA-esprimoj)

```
integer procedure fk (n, f);
   integer  n, f;
begin
  if n = 1
     then fk := f
     else fk := fk (n-1, n*f)
end;

print ( fk (6, 1))
```

```
mode intpar = struct (proc ref int md, proc int d);
proc (intpar, intpar) int fk =
     (intpar n, intpar f) int :
  if d of n = 1
     then d of f
     else fk ((ref int : (eraro; skip), int : d of n - 1),
              (ref int : (eraro; skip),
                   int : (d of n) * (d of f)))
  fi;

print ( fk ((ref int : (eraro, skip), int : 6),
            (ref int : (eraro, skip), int : 1)
          )   )
```

(ekzemplo 7: nom-transdono parametro de ALGOL 60)

```
type ref_real is access real;
   kopio : exception;
generic
   x : in ref_real;
   with function f return real;
function integralo (a, b, h : in real) return real;

function integralo (a, b, h : in real) return real is
   s : real := 0.0;
begin
  begin
    x.all := a;
    while b - x.all > h/3.0
    loop
      s := s + f();   x.all := x.all + h;
    end loop;
    return h*s;
  exception
    when NUMERIC_ERROR => raise kopio;
  end;
exception
  when kopio  =>
   PUT ("eraro en funkcio integralo");  NEW_LINE;
   PUT ("valoro de a ");  PUT (a);  NEW_LINE;
   PUT ("valoro de b ");  PUT (b);  NEW_LINE;
   PUT ("valoro de h ");  PUT (h);  NEW_LINE;
   PUT ("valoro de s ");  PUT (s);  NEW_LINE;
   raise kopio;
end  integralo;

x : ref_real := new real;
y : ref_real := new real;

function f1 return real is
begin
  begin
    return  x.all ** 2 - y.all ** 3;
  exception
    when NUMERIC_ERROR  =>
     PUT ("eraro en f1");  NEW_LINE;
     raise  kopio;
  end;
exception
  when  kopio  =>  raise  kopio;
end  f1;

function int1 is new integralo (x, f1);

function f2 return real is
begin
  return  int1 (0.0, 1.0, 1.0e-4);
exception
  when  kopio  =>  raise  kopio;
end f2;

function int2 is new integralo (y, f2);

PUT (int2 (0.0, 1.0, 1.0e-4));
```

(Ekzemplo 8: malkonstruanta storo-kopiilo en ADA)

```
program ekzemplo (output);
  label  8, 9;
  var    valoro : real;
  procedure kopio;
     begin   goto 8   end;
  function f (a : integer; procedure kopio) : real;
    label   8, 9;
    procedure kopio2;
       begin   goto 8   end;
  begin
    if a > 0 then f := a*f (a-1, kopio2)
             else  goto 8;
    goto 9;
   8 : writeln (' la funkcio f.');
       writeln (' valoro de a : ', a);
       write   (' alvokinto estas ');
       kopio;
   9 :
  end;
begin
  valoro := 0;   valoro := f (10, kopio);
  goto 9;
 8 : witeln (' la programo mem.');
 9 :
end.
```

(Ekzemplo 9: malkonstruanta storo-kopiilo en PASCAL)

programlingvoj

IUJ RIMARKOJ PRI LA ALGORITMA LINGVO ALGOL 60

Boris Gärtner
Schöttlstraße 12
D-8000 München 70

ŝlosilaj vortoj:

programlingvo ALGOL 60, nom-transdono, valor-transdono, celnomoj, malkompletaj specifoj, deskriptoroj, programlingvo ALGOL 68

Resumo:

Iuj specialaj kvalitoj de la programlingvo ALGOL 60 estas klarigata en terminaro de la lingvo ALGOL 68.

Some features of ALGOL 60 are explained in terms of ALGOL 68.

ALGOL 60, iom la plej fascinanta kreo en la mondo de algoritmaj lingvoj, perdigis post la apero de PASCAL grandan parton de ĝia signifo. Iom la studentoj sin klopodis kompreni la pli malfacilajn erojn de la lingvo; hodiaŭ ili atingas ofte nur tute malprofundan scion pri tiu ĉi iom pionira lingvo. Tial estas klarigataj iuj elementoj de la lingvo ALGOL 60 uzante la pli perfektajn esprimilojn de la lingvo ALGOL 68.

ALGOL 60 estas strukturata lingvo, kies plej grava strukturilo estas la proceduro. Proceduro povas resulti valoron, kies tipo estas aŭ real aŭ integer aŭ boolean. Parametroj de proceduroj estas la plej malfacila elemento de la lingvo. Ni konstatas:

1. Oni diferencigas nom-transdono (angle: call by name) kaj valor-transdono (angle: call by value).
2. Oni rajtas deklari tipon de parametro; deklaro estas deviga, se parametro estas nomata en la listo de valor-transdonaj parametroj.
3. La sekvaj deklaroj haveblas:
   real integer boolean
   real array integer array boolean array
   array servas kiel mallongigo de real array
   label switch
   procedure
   real procedure integer procedure boolean procedure
   string

4. Valor-transdono haveblas por la tipoj real, integer, boolean, real array, integer array, boolean array, label .
   Valor-transdono estas deklarata per valor-deklarilo, kiu sekvas al procedur-kapo kaj antaùas al tip-deklaroj de la parametroj.
5. ĉiuj deklaroj de parametroj, kiuj enhavas la vorton array aù la vorton procedure estas malkompletaj.

Pri valor-transdono la nomo de parametro estas kunligata kun valoro de aktuala parametro, kiu estas ellaborata nur unufoje pri la alvoko de la proceduro. Pri nom-transdono ĉiu ellaborado de formala parametro egalas al nova ellaborado de kies aktuala anstataùo en la ĉirkaùaĵo, en kiu la proceduro estis alvokata. Nomo de nom-transdona parametro uzeblas maldekstre de asignilo ":=" nur, se aktuala anstataùo estas esprimo, kiu nomigas variablon de tipo real, integer aù boolean.

Efektivigo de la parametroj real, integer, boolean

Estas uzata la sekvaj tipoj (de ALGOL 68)

```
mode realp = struct (proc ref real maldekstre, proc real dekstre),
     intp  = struct (proc ref int  maldekstre, proc int  dekstre),
     boolp = struct (proc ref bool maldekrste, proc bool dekstre);
```

Ekzemplo:

```
begin
  real procedure sumo (a, i, n);
    value n;   integer n;
    real a;    integer i;
  begin
    real s;   s := 0.0;
    for i := 1 step 1 until n do  s := s + a;
    sumo := s
  end;
  integer i, j;  array a, b [ 1 : 10 ] , d [ 1 : 10, 1 : 10 ];
  output (60, '('')', sumo (a [i ], i, 10), sumo (a[ i ]* b[ i ], i, 10),
          sumo (d[ i, i ], i, 10), sumo (sumo (d[ i, j ], i, 10), j, 10))
end;
```

Sekvas klarigo en ALGOL 68:

```
begin
  mode realp = struct (proc ref real maldekstre, proc real dekstre),
       intp  = struct (proc ref int  maldekrste, proc int  desktre);
  proc (realp, intp, intp) real sumo = (realp a, intp i, intp n) real :
  begin
    ref int valoro n = loc int := dekstre of n;  co valor-transdono co
    ref real s = loc real := 0.0;
    maldekstre of i := 1;
    while dekstre of i <= valoro n
    do
      s +:= dekstre of a;  maldekstre of i := dekstre of i + 1
    od;
    s
  end;
  ref [ ] real a = loc [ 1 : 10 ] real;
  ref [ ] real b = loc [ 1 : 10 ] real;
  ref [ , ] real d = loc [ 1 : 10, 1 : 10 ] real;
  print ( sumo ((ref real : a[ i ], real : a[ i ]),
                (ref int  : i     , int  : i ),
                (ref int  : (print ("erara asigno"); skip), int : 10)));
    co  sumo de la elementoj de la vico a  co
  print ( sumo ((ref real : (print ("erara asigno"); skip),
                      real : a [ i ] * b [ i ]),
                (ref int  : i     , int  : i),
                (ref int  : (print ("erara signo"); skip), int : 10)));
    co  sumo de la produtoj  a [ i ] * b [ i ]   co
  print ( sumo ((ref real : d [ i, i ], real : d [ i, i ]),
                (ref int  : i     , int  : i ),
                (ref int : (print ("erara asigno"); skip), int : 10)));
    co  sumo de la diagonalaj elementoj de la matrico d  co
  print (sumo ((ref real : (print ("erara asigno"); skip),
                real : sumo ((ref real : d [ i, j ], real : d [ i, j ]),
                             (ref int : i, int : i),
                             (ref int : (print ("erara asigno"); skip),
                              int : 10)),
               (ref int : j, int : j),
               (ref int : (print ("erara asigno"); skip), int : 10)))
    co  sumo de ĉiuj elementoj de la matrico d  co
end
```

Ni rimarkas, ke pri alvoko ne ekzistas malsameco inter nomtransdono kaj valortransdono. Oni povus pensi, ke valorparametro simple estas objekto de la tipo real, int aŭ bool. Se ni parolos pri procedur-valoraj parametroj, ni vidos, ke precize procedur-valoraj parametroj necesigas egalan traktadon de ambaŭ transdon-metodoj.

Se uzo de aktuala anstataŭo de parametro maldekstre de asignilo maleblas, la ero "maldekstre" de aktuala anstataŭo estas erar-komunikilo. Eraro tial estos dekovrata nur tiam, se ĝia ellaborado estas provata. Tial ALGOL-sistemo akiras trajton de interpretigaj sistemoj.

Efektivigo de la parametroj label kaj switch

Por celparametro uzeblas nomtransdono kaj valortransdono. Valortransdono diferencas al nomtransdono nur, se aktuala anstataŭo de celparametro estas celesprimo, kiu uzas malprivatajn de proceduro objekton. Se aktuala anstataŭo estas nur simpla celnomo, ne ekzistas diferenco inter valortransdono kaj nomtransdono.
La ekzemplo klarigas tion:

```
begin
  boolean a, b;
  switch s = l1, l2,
    if b then l3 else l4;
  procedure saltu (x);
    value x;  label x;
  begin
    a := b := false;  goto x
  end;
  a := b := true;
  saltu (if a then s[ 3 ] else l1);
  l1 : l2 : l3 : l4 : ;
end;
```

```
begin
  boolean a, b;
  switch s = l1, l2,
            if b then l3 else l4;
  procedure saltu (x);
    label x;
  begin
    a := b := false;   goto x
  end;
  a := b := true;
  saltu (if a then s[ 3 ] else l1);
  l1 : l2 : l3 : l4 : ;
end;
```

En la maldekstre skribata ekzemplo la celesprimo, kiu estas aktuala anstataŭo de la parametro "x", estas ellaborata post la aktivigo de la proceduro sed antaŭ la ellaborado de la procedur-instrukcioj. Resultas la celnomo "13", kiu estas storata. La

instrukcio goto x sekve egalas al goto *ℓ3* .

En la dekstre skribata ekzemplo la celesprimo, kiu estas aktuala anstataŭo de la parametro "x", estas ellaborata nur tiam, kiam estas ellaborata la instrukcio goto x. La valoroj de la malprivataj objektoj "a" kaj "b" nun estas false. Sekve la instrukcio goto x nun igas salton al celo *ℓ1* .

Celparametro posedas la sekvantan tipon (de ALGOL 68):

```
mode label = proc proc void;
```

Distribuilo *switch* simple estas vico de label-objektoj.
Nun ni kapablas novprogramigi la ekzemplojn uzante ALGOL 68:

```
begin
  mode label = proc proc void;
  [ 1 : 3 ] label s =
    (proc void : void : goto ℓ1, proc void : void : goto ℓ2,
     proc void : if b then void : goto ℓ3 else void : goto ℓ4 fi);
  ref bool a = loc bool := true,
           b = loc bool := true;
  proc (label) void saltu1 = (label x) void :    co nomtransdono co
  begin
    a := b := false;      x   co alvoko co
  end;
  proc (label) void saltu2 = (label x) void :    co valortransdono co
  begin
    ref proc void valoro x = loc proc void := x;
      co valoro de x nun estas ellaborita kaj stokita co
    a := b := false;     valoro x
  end;
    co alvokoj estas: co
  saltu1 (proc void : if a then s [ 3 ] else void : goto ℓ1 fi);
  saltu2 (proc void : if a then s [ 3 ] else void : goto ℓ1 fi);
  ℓ1 : skip; ℓ2 : skip; ℓ3 : skip; ℓ4 : skip
end
```

Se oni malpermesas valortransdonon de celnomoj, la tipo label povus esti simple proc void. (Fakte multaj ALGOL-60-sistemoj

malpermesas valortransdonon de celnomoj. Tial nomtransdono de celnomoj efektivigas pli simple.)

Efektivigo de celesprimoj kiel proceduroj solvas kaj problemon de senfinaj rekursivaj alvokoj en distribuiloj. Ekzemple

```
begin
  switch s := s[2], s[1];     goto   s[1]
end;
```

simple kaùzas troaĵon de la storo kaj tial finigas la ŝajne senfinan komputadon.

La raporto pri la lingvo ALGOL 60 permesas sensignaj entjeraj nombroj kiel celnomoj. Kutime oni malpermesas tion. La kialo por tiu malpermeso estas ligata kun la fakto, ke la parametro-deklaroj de proceduraj parametroj estas malkompletaj. Ni diros pri tio en la sekcio pri procedurvaloraj parametroj, sed nun ni donas ekzemplon:

```
begin
  procedure ellaboru (p);
     procedure p;   comment  p estas procedur-valora parametro;
  begin
    p (3)
  end;
  procedure p1 (celo);
     label celo;
    goto celo;
  procedure p2 (nombro);
     integer nombro;
  begin
    integer x;     x := nombro
  end;
    comment    sekvas du alvokoj ;
  ellaboru (p1);    comment  nun  "3"  estas celnomo ;
  ellaboru (p2);    comment  nun  "3"  estas entjera nombro;
end;
```

Se oni volus efektivigi tion, oni devus uzi la sekvantan tipon,

kiu kapablas representi celesprimojn kaj entjer-tipajn parametrojn:

```
mode intlab = struct (intp nombrero, label celero);
```

Uzo de tiu tipo estas la jena:

```
proc (proc (intlab) void) void ellaboru = (proc (intlab) void p) void :
  p (((ref int : (print ("erara asigno al konstanto"); skip),
       int : 3),                          co en ALGOL 68 entjera nombro co
      proc void : (void : goto 3)));      co ne pavas esti celnomo ! co
proc (intlab) void p1 = (intlab celo) void :
  celero of celo;
proc (intlab) void p2 = (intlab nombro) void :
begin
  ref int x = loc int := nombrero of nombro
end
```

La ĉiama transdono de ambaŭ tipoj de parametroj estas ĉiama malavantaĵo por tute malofta situacio. Tiam oni kutime malpermesas la uzon de sensignaj entjeraj nombroj kiel celnomoj.

Efektivigo de vicparametroj

Deklaro de vicparametro ĉiam estas malkomleta. Oni deklaras nur la tipon de la viceroj, sed la nombro de vic-indicoj restas malkonata. Same deklaro de procedurvaloraj parametroj ĉiam estas malkompleta, ĉar oni deklaras nur la tipon de la resulto. La nombro de procedurparametroj kaj iliaj tipoj restas malkonataj. Uzante ambaŭ malkompletaĵojn, oni povas skribi programojn, kiuj ne povas esti skribataj en la plejmulto de aliaj programlingvoj. La sekvanta ekzemplo ilustras tion.

```
begin
  procedure pp (a, p);
    array a;    real procedure p;
    comment  nombro de vic-parametroj de a, nombro de parametroj de
             p kaj iliaj tipoj estas malkonataj;
  begin
    output (60, '('')', p (a));
    comment  nun oni scias, ke p posedas unu parametron, kies tipo
             estas vico kun ankoraŭ malkonata nombro da vic-indicoj;
  end;
```

```
real procedure p1 (a);
  array a;  comment  la nombro de vic-indicoj ne estas deklarebla ;
begin
  integer i;    real s;    s := 0.0;
  for i := 1 step 1 until 10 do  s := s + a[ i ];
    comment  nun oni scias, ke a posedas unu indicon ;
  p1 := s
end;

real procedure p2 (a);
  array a;  comment  la nombro de vic-indicoj ne estas deklarebla ;
begin
  integer i, j;   real s;    s := 0,0;
  for i := 1 step 1 until 10 do
    for j := 1 step 1 until 10 do  s := s + a[ i, j ];
      comment  nun oni scias, ke a posedas du indicojn ;
  p2 := s
end;

array x[ 1 : 10 ], y[ 1 : 10, 1 : 10 ];

   comment  tie la vicoj x kaj y iĝas iliajn valorojn ;

pp (x, p1);   pp (y, p2);  comment  seneraraj alvokoj ;
comment    pp (x, p2)  kaj  pp (y, p1)  estas eraraj  ;
end;
```

Unua klarig-provo klarigas nur la ekzemplon mem, sed ne la ĝenerala metodo de la parametro-transdono por vic-tipoj.

```
begin
  mode  victipo = union (ref [ ] real, ref [ , ] real);
  proc (victipo, proc (victipo) real) void pp =
       (victipo a, proc (victipo) real p) void :
    print (("resulto estas ", p(a), newline));
   co  malsame al la skribita en ALGOL 60 proceduro pp, tiu ĉi proce-
       duro atendas proceduron p kun komplete deklarata tipo. Ni vidos,
       ke ni povus uzi la tipon
           union (proc (ref [ ] real) real, proc (ref [ , ] real) real)
       nur, se ĉiuj tip-provoj estus plenumata en tiu ĉi proceduro.
       Nun ni plenumas tip-provoj kiel eblas plimalfrue   co
```

```
proc (victipo) real p1 = (victipo a) real :
case a in      co  tiu estas la tipprova esprimilo   co
   (ref [ ] real a1) :
      begin
        ref real s = loc real := 0.0;
        for i from 1 by 1 to 10 do   s +:= a1 [ i ]   od;
        s    co  resulto  co
      end
  out     print (("erara victipo en proc p1", newline)); goto stop
esac;

proc (victipo) real p2 = (victipo a) real :
case a in
   (ref [ , ] real a2) :
      begin
        ref real s = loc real := 0.0;
        for i from 1 by 1 to 10 do
          for j from 1 by 1 to 10 do  s +:= a2 [ i, j ]  od  od;
        s
      end
  out     print (("erara victipo en proc p2", newline)); goto stop
esac;

ref [ ] real x = loc [ 1 : 10 ] real;
ref [ , ] real y = loc [ 1 : 10 , 1 : 10 ] real;

  co   la vicoj  x  kaj  y  iĝas iliajn valorojn   co

pp (x, p1);    pp (y, p2)
end
```

Estas videbla, ke la proceduroj "p1" kaj "p2" provas iliajn parametrojn. Dank al ĝeneraligo de la victipoj la proceduroj "p1" kaj "p2" estas de egala tipo. Alia ebleco estas la jena:

```
mode proctipo = union (proc (ref [ ] real) real, proc (ref [ , ] real) real);
proc (victipo, proctipo) void pp = (victipo a, proctipo p) void :
 print (case p in
        (proc (ref [ ] real) real p1) : case a in
                                          (ref [ ] real a1) : p1 (a1)
                                         out  eraro
                                        esac ,
```

```
        (proc (ref [ , ] real) real p2) :
          case a in
            (ref [ , ] real a2) : p2 (a2)
          out eraro
          esac
        esac );        co  fino de pp  co

proc (ref [ ] real) real p1 = (ref [ ] real a) real :
begin
  ref real s = loc real := 0.0;
  for i from 1 by 1 to 10 do  s +:= a [ i ]  od;   s
end;
```

k.t.p.

Bedaŭrinde la unuigilo *union* ne permesas ĝeneral-validan klarigon de la transdono de vicoj, ĉar la deklarlio *array* de la lingvo ALGOL 60 fakte nomigas unuigon de senfina nombro de vic-tipoj. La unuigilo de la lingvo ALGOL 68 tamen kapablas difini nur unuigtipajn objektojn, kiuj unuigas finan nombron de tipoj. ĝeneral-valida klarigo de la vic-transdono necesas difinon de vic-deskriptoro. Vic-deskriptoro konsistas el nombro da vic-indicoj, montrilo al indiclimoj kaj faktoroj de la retrovfunkcio kaj montrilo al la vico mem. Pli formala:

```
mode limoj = struct (ref [ ] int faktoroj, malsupre, supre);
mode realvico = struct (int dim,  co nombro da indicoj  co
                        ref limoj formo,  co vicindicoj, faktoroj co
                        ref [ ] real vcio);
mode intvico  = struct (int dim, ref limoj formo, ref [ ] int vico);
mode boolvico = struct (int dim, ref limoj formo, ref [ ] bool vico);
```

La deklaro

```
real array x, y [ 1 : 5 , 2 : 7 , 3 : 8 ];
```

kreas la sekvantan strukturon:

```
ref limoj f = loc limoj := (loc [ 1 : 3 ] int := (1, 5, 30),
                           loc [ 1 : 3 ] int := (1, 2, 3),
                           loc [ 1 : 3 ] int := (5, 7, 8));
realvico x = (3, f, loc [ 0 : 179 ] real),
       y = (3, f, loc [ 0 : 178 ] real);
```

La deklaritaj mallongige <t.e., kun komuna indiclimoj-priskribo) vicoj "x" kaj "y" uzas komunan substrukturon "f" de la tipo ref limoj .

La valoroj de la strukturero "faktoroj" de la strukturo "limoj" estas komputata tiel:

```
proc (ref limoj, int, proc (int) void) int  komputu faktoroj =
     (ref limoj d, int objektlongo, int dimensio, proc (int) void eraro)
 int :
begin
  ref [ ] int f = faktoroj of d,
              ms = malsupre of d,
               s = supre    of d;
  ref int p = loc int := objektlongo;
    co  objektlongo estas la stor-kvanto, kiu necesas por unu vicero  co
  for j from dimensio by -1 while j > 0
  do
    f [ j ] := p;
    p *:= (if  int longo = s [ j ] - ms [ j ] + 1;
               longo >= 1
               then longo
               else eraro (erara longo); skip
           fi)
  od;
  p   co  tio estas la stor-kvanto, kiu necesas por la tuta vico  co
end
```

La vicer-retrov-funkcio nun estas skalara produto:

$$s = \sum_{i=1}^{dim} faktoroj\ [\ i\ ]\ *\ (indico\ [\ i\ ]\ -\ malsupre\ [\ i\ ])$$

```
proc (int, [ ] int, ref limoj) int  vicero =
     (int dimensio, [ ] int  indicoj, ref limoj d) int :
begin
  ref [ ] int f  = faktoroj of d,
              ms = malsupre of d,
              s  = supre    of d;
  ref int p := loc int := 0;
```

```
    for i from dimensio by -1 to 1
    do
      p +:= (if int ii = indicoj [ i ], msi = ms [ i ], si = s [ i ];
                ii >= msi and ii <= si
                then ii - msi
                else eraro (erara indico); skip
             fi) * f [ i ]
    od;
    p   co  la loko de la vicero en la vico  co
  end;

  op elem = (realvico d, [ ] int indicoj) ref real :
     vico of d [ vicero (dim of d, indicoj, formo of d) ];

  op elem = (intvico  d, [ ] int indicoj) ref int :
     vico of d [ vicero (dim of d, indicoj, formo of d) ];

  op elem = (boolvico d, [ ] int indicoj) ref bool :
     vico of d [ vicero (dim of d, indicoj, formo of d) ];
```

Rigardu, ke ĉiuj resultoj, kiuj estas komputata dum la komputado, havas valorojn en la gamo

lwb vico of <deskritoro> : *upb vico of* <deskriptoro>

Tial la vicer-retrov-funkcio agas numere sekure (t.e., ke dum-komputadaj resultoj neniam igas tro grande)

Proceduraj parametroj

Parametro estas objekto de la tipo

```
mode parametro = union (realp, intp, boolp, label, [ ] label,
                        realvico, intvico, boolvico, proceduro,
                        realproc, intproc, boolproc, litervico);
```

Parametrolisto estas la jena strukturo:

```
mode parametrolisto = struct (int nombro, ref [ ] parametro p);
```

La proceduraj tipoj estas la jenaj:

```
mode realproc = proc (parametrolisto) real,
     intproc  = proc (parametrolisto) int,
     boolproc = proc (parametrolisto) bool,
     proceduro= proc (parametrolisto) void;
```

Pri alvoko de proceduro estas kreata objekto de la tipo *parametrolisto*. Alvokata proceduro unue ekzamenas nombron kaj tipon de ĝiaj parametroj. Samtempe kun tip-ekzamenado estas ellaborataj ĉiuj parametroj, por kiuj estas deklarata valor-transdono. Post la senerara finiĝo de la tip-ekzamenado komencas la ellaborado de la procedur-instrukcioj mem. La problemo de malkompletaj deklaroj estas solvata tiel, ke por malkompletan deklaron ankaŭ la tip-ekzamenado estas malkompleta. Aldonaj ekzamenadoj laŭbezone estos efikata de aliaj proceduroj.

Senparametra, resultellaboranta proceduro povas esti aktuala anstataŭo de formala parametro, kiu estas deklarata kiel nomaŭ valor-transdona parametro de tipo real, integer aŭ boolean. Tial senparametra, resultellaboranta proceduro pri alvoko estas traktata kiel nomtransdona simpla parametro. La ago de la alvokato estas la jena:

Se estas konata, ke resultellaboranta procedurvalora parametro atendas parametrojn, tipo de ĝia aktuala anstataŭo nepre devas esti realproc, intproc aŭ boolproc (laŭ tipo de la resulto.)

Se estas konata, ke resultellaboranta procedurvalora parametro estas uzata sen parametroj, tipo de ĝia aktuala anstataŭo nepre devas esti realp, intp aŭ boolp (laŭ tipo de la resulto.)

se ne estas konata, ĉu parametro estas uzata kun aŭ sen parametroj, tipo de ĝia aktuala anstataŭo devas esti
union (intp, intproc), union (realp, realproc) aŭ
union (boolp, boolproc) laŭ tipo de la resulto.

La sekvantaj ekzempletoj ilustas tiujn tri eblecojn:

```
procedure f (p);                    proceduro f = (parametrolisto pl) void :
   real procedure p;                  if nombro of pl = 1
begin                                   then case p of pl [ 1 ] in
   x := p                                        (realp pp) : x := dekstre of pp
end;                                             out  parametreraro (1)
                                            esac
 (senparametra)                        else eraro (parametronombro)  fi
```

```
procedure f (p);
   real procedure p;
begin   x := p (3.14)    end;
```

egalas al

```
realproc f = (parametrolisto pl) void :
  if nombro of pl = 1
    then case p of pl [ 1 ] in
          (realproc pp) :
            x := pp ((1, loc [ 1 : 1 ] parametro
                        := (ref real : (eraro, skip), real : 3.14)))
          out parametroeraro (1)
         esac
    else eraro (parametronombro)
  fi
```

```
procedure f (p, p1);
   procedure p;
   real procedure p1;
begin  p (p1)    end;
```

egalas al

```
realproc f = (parametrolisto pl) void :
  if nombro of pl = 2
    then case p of pl [ 1 ] in          co  unua parametro  co
          (proceduro pp) :
            case p of pl [ 2 ] in       co  dua  parametro  co
             (union (realp, realproc)) :
               pp ((1, ref [ ] parametro (p of pl [ 2 ])))
             out  parametroeraro (2)
            esac
          out  parametroeraro (1)
         esac
    else  eraro (parametroeraro)
  fi
```

Valortransdono estas efektivigata jene:

```
  procedure p (a);
     value a;   array a;
  begin   comment instrukcioj;   end;
```

transformigas al

```
realproc p = (parametrolisto pl) void :
  if nombro of pl = 1
    then case p of pl [ 1 ] in
          (realvico rv) :
           begin
             realvico a = (dim of rv, formo of rv,
                           loc [ 1 : upb vico of rv ] real :=
                              vico of rv);
             co   nun ĉiuj instrukciojreferas al a, ne al rv  co
             skip
           end
         out  parametroeraro (1)
         esac
    else  eraro (parametronombro)
  fi
```

Liter-vicoj

Restas la problemo, kiel estas efiktivigataj liter-vicoj. Ili estas objektoj de la tipo

```
mode litervico = ref [ ] char;
```

Ilia uzebleco estas tute limigata; ili uzeblas nur kiel aktualaj parametroj de elig-proceduroj kaj kiel parametroj de proceduroj, kiuj ne estas skribataj en la lingvo Algol 60 mem. Valor-transdono ne haveblas.

ĉu tiu ĉi metodo taŭgas? Sendube, se proceduro nur ellaboras sufiĉe grandan agon. Pri proceduroj, kiuj ellaboras nur etan agon kaj alvokiĝas ofte, la objekt-ekzamenado prenigas grandan parton de la komputtempo. Per tiu rimarko estas montrata la ĉefa limigo de la lingvo Algol 60. Malgraŭ tio Algol 60 ĝis hodiaŭ estas promesplena lingvo. La fakto, ke la paramdeklaroj estas malkompletaj, ege simpligas la tradukilon. Fakte tradukilo por la lingvo Algol 60 estas multe pli mallonga ol tradukilo por iu ajn lingvo kun kompletaj parametro-deklaroj. Aliaflanke la malkompletaĵo de ĝiaj deklaroj ne estas ligata kun limigoj, kiuj ekzistas en multaj "pli modernaj" lingvoj.

La klarigita metodo estas la sola, kiu estas tiel ĝenerala, ke oni kapablas efektivigi procedur-bibliotekojn. Se tradukilo estas konstruata nur por la traduko de kompletaj programoj, kiuj ne enhavas rilatojn al procedur-bibliotekoj, oni povus provi plibonigon de la klarigita metodo. Tiam programplibonigo esence estas tip-simpligo. Oni strebas trovi pli simplaj anstataŭigoj por la tipoj intp, realp, boolp kaj por la proceduraj objektoj. ĉiu aparta programplibonigа klopodo necesas la eble plurfojan tralegon de la tuta program-teksto. Programplibonigo resultas en esenca plibonigo nur tiam, se ĝi sukcesas forigi la unuig-tipojn kaj la ligatajn kun ili ekzamen-frazojn. Tio eblas nur,se la esprim-kapabloj de la lingvo ne estas elĉerpataj. Volortransdono de simplaj tipoj (real, integer, boolean) ofertas la plej promesajn eblecojn por programplibonigo. Tial kutime la klopodoj de programplibonigo estas koncentrataj al simplaj valortransdonaj parametroj.

LITERATURO

Randell, B.; Russell, L. J.:
Algol 60 Implementation
1964 Academic Press London New York

Dahlstrand, I.; Naur, P.:
Integers as Labels
Algol Bulletion No. 10, October 1960
Regnecentralen Copenhagen.

Grau, A. A.; Hill, U.; Langmaack, H.:
Translation of Algol 60
1967 Springer Berlin Heidelberg New York

Backus, J. W. kaj aliaj:
Revised Report on the Algorithmic Language ALGOL 60
Numerische Mathematik 4, 420 - 453 (1963)

esperantlingvo traduko:
Revisiita raporto pri la algoritma lingvo Algol 60
1983 Budapesto, Hungara Esperanto-Asocio

van Wijngaarden, Aadrian, kaj aliaj:
Revised Report on the Algorithmic Language ALGOL 68
1976 Springer Berlin Heidelberg New York

Defenda kaj dinamika analizo.

fako : analizo sistema Belgio, Marc Vanden Bempt

Resumo en Esperanto.

Pere de modelo konsistanta el tri difinitaj aroj la principoj de defenda kaj dinamika analizo estas klarigataj. Tiuj principoj gvidu la analiziston ĉe sia decido ĝis kiu nivelo la disvolvenda informsistemo estu aùtomatigata.
La principoj validas sendepende de la uzata analizometodo.

Resumo en la nederlanda / nederlandstalige samenvatting.

Via een model met drie verzamelingen (zie schema in punt 2) worden de principes van defensieve en dynamische analyse gegeven.

Model :
- A = gevallen waarover men eensgezind akkoord is van ze in het te ontwerpen systeem te voorzien : 'probleemgebied' ;
- B = twijfelgevallen, of niet besproken mogelijkheden ;
- C = expliciet of evident overbodige gevallen.

Defensieve analyse :
1. het systeem test op gevallen uit C en kan er bijgevolg op steunen dat deze niet voorkomen.
2. het systeem laat alleen gevallen uit A door als correct, i.p.v. alleen C als fout aan te duiden.

Dynamische analyse :
3. hoe groter A, des te breder is het systeem en des te hoger de voldaanheid der gebruikers.
4. het systeem moet zo weinig mogelijk op de beperkingen in A steunen : uitbreidbaarheid.
5. de 3 verzamelingen moeten duidelijk afgelijnd zijn.

Defenda kaj dinamika analizo.

1. Enkonduko.

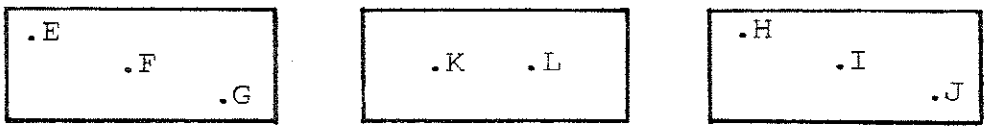
Ĉiu metodo por disvolvi informsistemojn, ĉu ĝenerale akceptita ĉu interne de la kompanio ellaborita, supozas analizofazon.
Iuj metodoj preskribas norman formon por la dokumentoj liverendaj post ĉi tiu fazo de la analizisto, aliaj preskribas certajn analizadoregulojn kiuj ekzemple fiksas devigan sinsekvon inter donitaĵo- kaj procedoanalizoj, ankoraŭ aliaj enhavas aŭtomatajn helpilojn kiel programprototipilo, donitaĵvortaro, k.a..
Ĉiuj metodoj povas ege diferenci inter si rilate al tiuj ĉi facetoj. Kiuj ajn estas la diferencoj kaj kiom ajn profunde la diversaj metodoj preskribas regulojn, unu econ ili ĉiuj havas : ili lasas la analiziston tre libera pri la demando ĝis kiu nivelo li devos aŭtomatigi la solvon por la informproblemo. La respondon al ĉi tiu demando li devas fiksi por ĉiu sistemo denove. Plej ofte estas la analizisto kiu decidas pri tiu nivelo de aŭtomatigo, pli-malpli en dialogo kun la finuzontoj de la disvolvenda informsistemo.

Ĉe sia decido la analizisto povos esti gvidata de la fundamentaj principoj de la problemo, kiujn ni ĉi sube, , kun ekzemploj, klarigos.

2. Modelo.

Disvolvenda sistemo ĉiam estas bazita sur, pli-malpli eksplicitaj, postuloj de ĝiaj uzontoj. Tiuj lastaj ne ofte priskribas ĉiujn eblecojn, kiom ajn klaraj ili estas.
Supozu ke E, F kaj G estas eblecoj kiujn la uzontoj volas retrovi en la sistemo.

Supozu ke H, I kaj J estas eblecoj kiuj laù la uzontoj certe ne troviĝu en la sistemo, respektive eblecoj pri kiuj ili tute ne parolas sed kiuj evidente ne apartenas al la esplorenda teritorio.
Supozu fine ke K kaj L estas eblecoj pri kiuj la uzontoj ne kuraĝas decidi, pri kiuj estas dubo; eblecoj pri kiuj li fi ne diras ke ili ne necesas en la sistemo, sed pri kiuj li tamen iome dubas; ankaù eblecoj pri kiuj estas (implicite) parolata, sed pri kiuj oni ne eksplicite decidas aù konsentas.



A B C

Ni do difinas la arojn :
A = eblecoj kiuj estas aùtomatigendaj : 'problemteritorio';
B = dubeblecoj, ne aùtomatigendaj;
C = certe ne aùtomatigendaj eblecoj.

Grava postulo estas ke la unuigo de tiuj 3 aroj entenu ĉiujn eblajn (kaj ŝajne neeblajn) kazojn.

3. Defenda analizo.

Ni uzos la ĉi supran modelon por klarigi "defendan projekta don" aù "defendan analizon" :

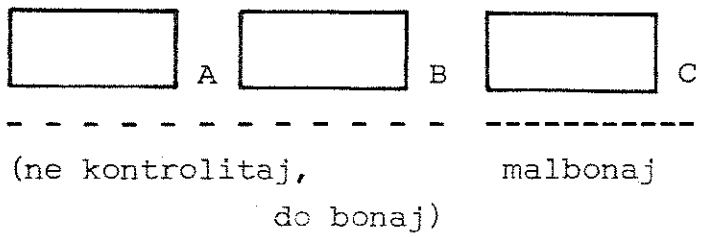
- PRINCIPO 1 : en la sistemo ni kontrolas la kazojn en C : se iu uzanto tamen prezentas tian kazon, la sistemo rifuzu ĝin kaj avertu la uzanton pri tio.
  Se poste, eble post jaroj, la uzanto modifos siajn postulojn kaj tamen volas aldoni kelkajn eblecojn el C al la sistemo (en aliaj vortoj : li volas pligrandigi la aron A kaj malpligrandigi aron B), tiam ni devos fari eksplicitan agon.
  Eblas fakte ke partoj de la sistemo uzas la fakton ke la kazoj el C ne eblis. Ĉar nun ni volos ke la sistemo akcep

tu eblecojn el C, ni devos ĝin entute repripensi kaj modi fi kie necesas.
La averto kiun la sistemo prezentas atentigas nin pri tio ke verŝajne la sistemo ie uzas la nomitan supozon.

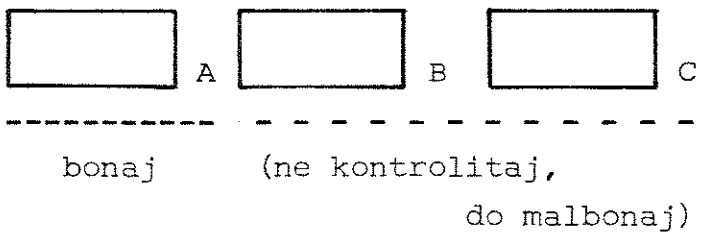
- PRINCIPO 2 : plej oftajn erarojn kaùzas la kazoj el aro B. La uzanto ne povas klare indiki pri kiuj kazoj precize temas. Kelkaj partoj de la sistemo toleras ilin, permesas ilin, en aliaj oni supozas ke ili ne povas prezentiĝi sen pli frua rifuzo flanke de la sistemo mem : en la plej multaj sistemoj tamen oni simple ne kontrolas ilin.
Tio ne estu permesata.
Do, anstataù simple kontroli ĉu kazoj el C ne prezentiĝas (sekve A kaj B bonaj), oni kontrolu ĉu nur prezentiĝas kazoj el A (sekve B kaj C malbonaj). Nur tiam ni povos esti certaj ke la sistemo taùge funkciu ne nur por A kaj C, sed same por kazoj el B.

Skeme :

- ne kontrolu :



- jes kontrolu:



Tio estas la vera senco de defenda analizo : ni volas eviti ke nia sistemo erare prilaboru la kazojn el B, dum ke la uzanto kredas ke kelkaj kazoj el B tamen estos senerare akceptitaj : se tiujn li prezentas al la sistemo, tiu lasta lin haltigos ...

Se post tio tamen li volas akceptigi tiujn specifajn eblecojn (el B, aù eĉ C), do transigi ilin al A, tiam la analizisto devos denove trarigardi ĉiujn facetojn de la sistemo por kontroli ĉu la deziratajn kazoj adaptiĝas en la konstruita tutaĵo. Nur tiam li povos forigi la kontrolon.

4. Ekzemploj.

Ekzemplo A :
Post impostkalkulo por produkto oni enkondukas kontrolon : 0 < imposto ≤ fiksita procento de la prezo de la produkto (iu procento < 100). Negativa imposto estas kazo el C : ĝi plej verŝajne neniam eblos; imposto egala al 0 aù pli granda ol la fiksita procento de la produktprezo estas kazoj el B : la uzantoj (aù la analizisto ...) supozas ke ili neniam okazos. Pro la menciita testo aliaj partoj de la informsistemo rajtas supozi la testitan kondiĉon.
Se poste oni volas permesi ekzemple nulan imposton, tiam oni devos eksplicite kontroli ĉu la sistemo neniel uzas la supozon, ekz. ĉe divido per la imposto.

Ekzemplo B :
En librotenosistemo uzanta fremdajn valutojn estis antaùvidata ke oni povu registri ricevendajn aù pagendajn sumojn , laù kiu ajn valuto ; aliflanke eblis registri la veran ricevon aù pagon, sed nur laù la sama valuto aù en belgaj frankoj (tiu ĉi sistemo funkciis en la belga asekurkompanio ABB en Leuven).
Estis eksplicite ne antaùvidita kaj do neebligita , ke oni registru pagojn/ricevojn laù alia valuto ol la origina aù BF. La sistemo uzis tiun supozon ĉe la perioda rekalkulo de la sumoj al belgaj frankoj.
Ni poste reuzos ĉi tiun ekzemplon.

5. Dinamika analizo.

Ni uzu la saman modelon kiel ĉe la klarigo de defenda anali

zo por klarigi dinamikan analizon :

- PRINCIPO 3 : ju pli da kazoj oni, jam dum la analizado , lokigis en la aro A de permesataj eblecoj, des pli larĝa estos la konstruita sistemo, des malpli da eblecoj (=plej ofte esceptoj) poste devos esti aldonataj, des pli granda estos la kontenteco de la uzantoj.

- PRINCIPO 4 : la sistemo tamen devos esti tiel supla ke ĝi ne tro sin apogu sur la limigoj kiuj, kiom ajn larĝa tiu aro estu, tamen troviĝas en A. Devas nome ebli sen multe da peno aldoni ankaŭ kazojn el C kaj precipe el B al la sistemo.
  Fakte ĝi devus ekde la komenco bone funkcii por ĉiuj kazoj el A kaj B (se tio estas finance pravigebla ...), eĉ se ni nenion el B permesos.

- PRINCIPO 5 : necesas ke ne nur aro A, sed ankaŭ B kaj C , kiom eble plej klare estu priskribitaj dum la sistemprojektado. Inter ili aro B devas esti kiom eble plej malvasta (teorie ĝi eĉ estu malplena).
  Jen lasta, sed plej grava principo.

Oni jes bone konsciu ke dinamikeco ofte estas inverse proporcia al simpleco kaj malmultekosteco. La analizisto respondecas pri la elekto de plej alta kvalito/kosto-proporcio. Por tio li povas sin apogi preskaŭ nur sur sia sperto, konsiderante tamen la ĉi suprajn 5 principojn.

6. Ekzemploj.

Ekzemplo A :
Rekonsideru ekzemplon B ĉe defenda analizo.

En la momento kiam la sistemo estis ellaborita, la nomita supozo estis evidenta : la kompanio ĉion pagis kaj ricevis pere de ĝirkontoj funkciantaj en belgaj frankoj. En aliaj

vortoj : por ĉiu pago aŭ ricevo okazanta laŭ fremda valuto la ĝirkonta saldo modifiĝis per la sumo en belgaj frankoj kiu konformas al la origina sumo laŭ la kurzo de la pagdato.

Du jarojn poste la kompanio malfermis ĉe specifaj bankoj ankaŭ kelkajn ĝirkontojn en specifaj fremdaj valutoj (germanaj markoj, kanadaj kaj usonaj dolaroj, ...). Dependo de diversaj kondiĉoj ekde tiam oni pagas ankaŭ en fremdaj valutoj. Ekzemple : sumo pagenda en francaj frankoj antaŭe ĉiam estis pagata el la BF-konto, do kun valoro en BF. Sed nun oni povas ĝin pagi el la konto kun la plej alta saldo aŭ kun la plej favora tagkurzo, ĉu en belgaj aŭ francaj frankoj aŭ ekzemple en kanadaj dolaroj.

La sistemo ne antaŭvidis ĉi tiun modifon (kazo el C): ne sufiĉe dinamika laŭ principo 3. Sed tio ne estis eraro de la analizisto : momente de la analizado la kazo estis tre malverŝajna kaj akcepti ĝin en la aro A malbone estus influanta la kvalito/kosto-proporcion. Aliflanke la analizo estis dinamika laŭ principo 5, ĉar la menciita ebleco eksplicite troviĝis en aro C, ne en B.
Same cetere ĝi estis dinamika laŭ principo 4, ĉar la modifo ne estis malfacila.

Ekzemplo B :
En la sama librotenosistemo oni povis sur la ekranon voki konton indikante la plenan nomon de la konto (kompreneble tio same eblis pere de la kontnumero ...).
Poste la uzantoj petis vokeblecon pere de nur la unuaj dek kvin karakteroj (el kvardek) de la nomo : tio estis tre facila modifo. Estis do kazo el B sen tro da malfacilaĵoj al A transigebla, sed tamen origine ne antaŭvidita.

Se la uzantoj ankaŭ petos la eblecon je serĉado pere de la unuaj n karakteroj de la nomo (n libera inter 0 kaj 40), tiam ankaŭ tio eblos sen multaj problemoj : nova B-kazo transigenda al A.

7. Konkludo.

Estas saĝa ago de la analizisto, kio ajn estu la uzata analizometodo, ke li zorgu por klara difino de la tri aroj A , B kaj C, el kiuj la aro B iĝu kiom eble plej malvasta, pere de daŭra celkonscia demandado al la finuzantoj de la projekt tenda sistemo.

Biografio.

Marc VANDEN BEMPT naskiĝis en 1956; diplomiĝis pri matematiko en 1977, pri informadiko en 1978; de tiam estas analizisto ĉe A.B.B. (Asekurkompanio de la belga 'Boerenbond'), en Leuven, Belgio.

MAGYAR TUDOMÁNYOS AKADÉMIA
KÖZPONTI FIZIKAI
KUTATÓINTÉZETE

Cím: H-1525.
Budapest Pf. 49.

Telefon:
166-209

Telex:
22-4289

HUNGARA SCIENCA AKADEMIO
CENTRA FIZIKA
ESPLORA INSTITUTO

Adreso: H-1525.
Budapešto Pf. 49.

Telefono:
166-209

Telekso:
22-4289

KFKI evoluigas la megamini-komputilon T P A – 1 1 / 4 4 0 – taŭgan por sciencaj, teknikaj komputadoj, interaktivaj grafikaj sistemoj, realtempa prilaboro laboratoria, medicina kaj de industria procezo-regado.

KFKI partoprenas en la ellaboro de la programada lingvo A D A – – utiliganta la komputistikajn esprolojn kaj spertojn de la pasintaj jaroj.

KFKI agadas sur la kampo de d a t u m b a z o j – celante plifaciligi la rilaton de la uzanto al la datumbazo.

KFKI ellaboras i n d u s t r i a j n p r o c e z r e g a d a j n kaj a d m i n i s t r a j n s i s t e m o j n surbaze de TPA–CAMAC.

TABELO DE LA ENHAVO.

Listo de landaj asocioj

Aperas en tiu listo nur la landoj por kiuj UEA registris pli ol 50 delegitoj en 1984.

Argentino. Argentina Esperanto-Ligo Casilla de Correo 17, Sucursal 53, RA-1453 Buenos Aires.
Aŭstralio. Aŭstralia Esperanto-Asocio P.O.Box 48, Jamison Centre, ACT 2614.
Belgio. Belga Esperanto-Federacio Montoyerstr. 37 b.24, B-1040 Bruxelles.
Brazilo. Kultura Kooperativo de Esperantistoj Av. Treze de Maio 47, Sobreloja 208, BR-20031 Rio de Janeiro (RJ).
Britio. Esperanto-Asocio de Britio, Esperanto-Centro 140 Holland Park Avenue, London W11 4UF.
Bulgario. Bulgara Esperantista Asocio Bul. Ĥristo Botev 97 BG-1303 Sofia.
Ĉeĥoslovakio. Ĉeĥa Esperanto-Asocio Jilská 10, CS-11001 Praha 1.
Danlando. Dana Esperanto-Asocio Tordenskjoldsgade 2-II, DK-8200 Århus N.
Finnlando. Esperanto-Asocio de Finnlando pf.48, SF-00131 Helsinki 13.
Francio. Unuiĝo Franca por Esperanto 4bis, rue de la Cerisaie F-75004 Paris.
Germanio, F.R. Germana Esperanto-Asocio Lambertstrasse 12 D-6430 Bad Hersfeld.
Hispanio. Hispana Esperanto-Federacio Apartado 119, Valladolid
Hungario. Hungara Esperanto-Asocio (HEA) pf.193, H-1368 Budapest.
Israelo. Esperanto-Ligo en Israelo P.O.K. 1289, IL-61012 Tel-Aviv.
Italio. Itala Esperanto-Federacio Via Volloresi 38, I-20143 Milano.
Japanio. Japana Esperanto-Instituto Waseda-mati 12-3, Sinzyuku-ku Tôkyô-to 162.
Jugoslavio. Jugoslavia Esperanto-Ligo (JEL) Terazije 42 YU-11000 Beograd.
Nederlando. Nederlanda Esperanto-Asocio Riouwstraat 172, 2585 HW Den Haag.
Norvegio. Norvega Esperantista Ligo (NEL) Olaf Schous vei 18, Oslo 5.
Nov-Zelando. Nov-Zelanda Esperanto-Asocio (NZEA) P.O.Box 330 Wellington 1.
Pollando. Pola Esperanto-Asocio ul. Jasna 6, PL-00-013 Warszawa.
Sovet-Unio. Asocio de Sovetiaj Esperantistoj Prosp. Kalinina 14, SSOD, SU-103009 Moskva.
Svedio. Sveda Esperanto-Federacio (SEF) Brunnsgatan 21, S-11138 Stockholm.
Svislando. Svisa Esperanto-Societo (SES) Schorenstr. 32 CH-4900 Langenthal.
Usono. Esperanto-Ligo por Norda Ameriko (ELNA) P.O:Box 1129, El Cerrito, CA 94530.