Univerzitet u Beogradu
Matematički fakultet - Beograd

Jozef J. Kratica

PARALELIZACIJA FUNKCIONALNIH PROGRAMSKIH JEZIKA I IMPLEMENTACIJA NA TRANSPJUTERSKIM SISTEMIMA

Magistarski rad

B e o g r a d
1994.

Mentor: prof. dr Dušan Tošić
Matematički fakultet - Beograd

Članovi komisije: prof. dr Slaviša Prešić
Matematički fakultet - Beograd

prof. dr Velimir Simonović
Mašinski fakultet - Beograd

prof. dr Žarko Mijajlović
Matematički fakultet - Beograd

Datum odbrane : _____

Paralelizacija funkcionalnih programskih jezika i implementacija na transpjuterskim sistemima

U radu je razmatrana paralelizacija funkcionalnih programskih jezika i implementacija takvog koncepta na transpjuterskim sistemima. Prikazana je realizacija interpretera za programski jezik LISP. Realizovane funkcije su standardne i zajedničke za većinu dijalekata LISP-a, sa mogućnošću jednostavne nadogradnje. Interpreter za LISP je implementiran na višeprocesorskom računaru sa prosledjivanjem poruka, arhitekture binarnog drveta.

Ključne reči:

funkcionalni

paralelni

transpjuteri

LISP

S-izraz

interpreteri

binarno drvo

višeprocesorski

prosledjivanje poruka

Paralelization of functional programming languages and implementation to transputer systems

In this paper we examine the paralelization of functional programming languages and implementation of that concept to transputer systems. LISP interpreter implemented here contains standard functions, common for all LISP versions. Adding new functions to the interpreter is easy. The interpreter is implemented for message passing multiprocessors, with binary tree architecture.

Key words:

functional

parallel

transputer

LISP

S-expression

interpreter

binary tree

multiprocessor

Message Passing

Sadržaj

Uspomeni na mog tragično
preminulog oca

Predgovor

Rad se sastoji iz pet poglavlja.

U uvodnom poglavlju su date osnovne činjenice o funkcionalnim programskim jezicima, paralelnim arhitekturama, i karakteristikama transpjuterskih sistema.

Drugo poglavlje sadrži činjenice o S-izrazima kao teorijskoj osnovi programskog jezika LISP.

Implementacija programskog jezika LISP na jednoprocesorskim računarima (PC) je data u trećem poglavlju.

Najvažniji deo rada je implementacija LISP-a na višeprocesorskim (transpjuterskim) sistemima. Ona je detaljno opisana u četvrtom poglavlju, uz prednosti i mane koje karakterišu ovaj metod implementacije.

Poslednje poglavlje sadrži zaključak, mogućnosti daljeg poboljšanja, i mogućnosti i značaj daljeg istraživanja u ovoj oblasti.

Spisak literature koja je korišćena nalazi se na kraju rada.

Interpreter za jednoprocesorske računare (iz trećeg poglavlja) testiran je na PC računaru, a interpreter iz petog poglavlja na transpjuterskom sistemu sa 17 transpjutera. Oba interpretera su testirana uz pomoć više test primera (na LISP-u), od kojih su neki i navedeni u radu.

Zahvaljujem se:

Sestri Lidi na pomoći u slaganju teksta, a takodje i majci Lidi na pruženoj podršci.

Mentoru Prof. dr Dušanu Tošiću na izuzetnom strpljenju i veoma korisnim stručnim savetima.

Članovima Komisije Prof. dr Slaviši Prešiću, Prof dr Velimiru Simonoviću i Prof. dr Žarku Mijajloviću na pažljivom čitanju rukopisa i korisnim savetima u izradi ovog rada.

Matematičkom Institutu u Beogradu, na korišćenju njihovog transpjuterskog sistema, pri realizaciji paralelnog LISP-a.

Kolegama mr Slobodanu Radojeviću, Milanu Vugdeliji, Draganu Uroševiću i mr Zdravku Stojanoviću na utrošenom vremenu i iskrenom prijateljstvu.

Beograd, april 1994. godine

Kandidat:

Jozef Kratica, dipl. mat.

Oznake i konvencije:

Da bi se izbegle dvosmislenosti, u radu se implementirane funkcije programskog jezika LISP, nazivaju funkcije, a programske celine koje služe za implementaciju interpretera (funkcije u programskom jeziku C) se nazivaju procedure.

Zaštićena imena:

1. Uvod

1.1. Funkcionalni programski jezici

U proceduralnim programskim jezicima (Pascal, C, Fortran, ...) osnovna (najmanja) konstrukcija je naredba, a u funkcionalnim je to funkcija. U funkcionalnim programskim jezicima naredbe i operatori se realizuju funkcijama.

Funkcionalne programske jezike karakterišu:

❶ Definicija funkcije, odnosno dodeljivanje tela funkcije (izraza koji daje neku vrednost) imenu funkcije. Pri definiciji funkcije koriste se brojni izrazi u kojima učestvuju konstante (brojne ili simboličke), promenljive i druge definisane funkcije.

❷ Izračunavanje vrednosti funkcije (poziv funkcije).

Treba imati u vidu i kratko zapisivanje "složenijih programa", koje potpuno odgovara matematičkom zapisu definicije tih funkcija. Implicitno koristimo rekurziju što skraćuje zapis.

Umesto termina "napisati program" pogodniji je termin "definisati funkciju", a umesto termina "izvršavanje programa" termin "izračunavanje izraza". Funkcije imaju ravnopravan tretman kao i podaci, tako da je moguće korišćenje funkcija kao podataka, a takodje i podataka kao funkcija (izvršavanje). Funkcija, koja koristi druge funkcije kao podatke, u literaturi se naziva funkcija višeg nivoa (higher order function).

Prednosti funkcionalnih jezika, u odnosu na proceduralne programske jezike su:

❶ Precizni su i jasni.

❷ Kraći su od programa na proceduralnim programskim jezicima.

❸ Manje se greši.

❹ Otkrivanje i ispravljanje grešaka je olakšano.

❺Jednostavno proširivanje konstrukcija jezika novim ugradjenim i korisničkim funkcijama.

❻ Lako definisanje rekurzivno definisanih funkcija.

❼ Implicitno zadat raspored izračunavanja, različit od proceduralnih jezika gde se eksplicitno zadaje niz naredbi koje se izvršavaju u tačno zadatom redosledu.

❽ Nepostojanje sporednih efekata (side effect), tj. isti izraz u istoj okolini daje istu vrednost, odnosno funkcija pozvana sa istim argumentima daje uvek istu vrednost.

❾ Velike mogućnosti paralelizacije.

❿ Mogućnost formalnog dokazivanja ispravnosti programa.

Pokazalo se da striktno pridržavanje funkcionalnog stila programiranja, prouzrokuje znatno uže izražajne osobine programskog jezika. Zbog toga se često javljaju i nefunkcionalni elementi.

Najpoznatiji predstavnik funkcionalnih programskih jezika je LISP (čiji dijalekti često sadrže i proceduralne elemente), a tu su i FP, SASL, Miranda, KRC, Haskell, ML, Iswim (detaljnije opise funkcionalnih programskih jezika videti u [3] i [4]).

1.2. Programski jezik LISP

LISP je prvi i najznačajniji funkcionalni programski jezik. Nastao je početkom 60-tih godina (1958.-1963.), a projektovao ga je John McCarthy, vođa grupe za veštačku inteligenciju na MIT-u. Prvobitno je bio namenjen za simboličke operacije sa listama (LISt Processing).

LISP nikada nije bio zvanično standardizovan (za razliku od Fortrana, Cobola, C-a, Pascal-a . . .), što je prouzrokovalo postojanje velikog broja dijalekata. Najpoznatije verzije LISP-a su:

XLISP, MuLISP, Franz LISP, LispKit-LISP , MACLISP, Zeta LISP, BBN-LISP, InterLISP, Standard-LISP, NIL (New Implementation of Lisp), Stanford LISP, Common LISP, itd.

Primena LISP-a je u veštačkoj inteligenciji (procenjuje se da je većina programa za veštačku inteligenciju napisana na LISP-u), simboličkom računanju, ... Zbog neusaglašenosti različitih verzija bilo je teško prenositi izvorni kod. Imajući u vidu i "težak" zapis aritmetičkih izraza, ipak nije doživeo popularnost Pascal-a ili C-a.

Common LISP je definisan 1981. kao nezvanični standard. Počeo je rad na zvaničnom standardu, koji je baziran na Common LISP-u, ali još uvek nije završen.

Postojanje zvaničnog standarda omogućava lakši prenos izvornog koda sa nekog računara na drugi, odnosno, sa jedne verzije LISP-a na drugu.

Implementacija u ovom radu je ograničena na onaj deo LISP-a koji je zajednički za većinu verzija, tako da je moguća nadogradnja do neke druge verzije, dodavanjem imena i definicija ugradjenih funkcija.

1.3. Višeprocesorske arhitekture

Moć računarske tehnike je do sada rasla veoma brzo. To se postizalo boljim tehničkim rešenjima pri kojima jedan procesor bržim radom poboljšava performanse čitavog sistema. U poslednje vreme glavnom procesoru u tome pomaže i po nekoliko specijalizovanih procesora od kojih svaki preuzima neki deo poslova (disk kontroleri, video procesori, ...). Medjutim, stiglo se vrlo blizu tehničkih granica. Stoga se pronalaze nova rešenja.

Rešenje se traži u višeprocesorskim računarima i paralelnom izvršavanju. Postoje različiti aspekti (nivoi) paralelizma.

Jedno rešenje je u preklopljenom pripremanju i izvršavanju instrukcija procesora (pipelining), gde je primenjen paralelizam niskog nivoa. Karakteristika ovakvog rešenja je da se procesorska instrukcija deli na više delova. Dati delovi instrukcija se izvršavaju paralelno. Za vreme izvršavanja jednog dela instrukcije, vrši se priprema drugog dela instrukcije, a prosledjivanje rezultata treće instrukcije. Nove generacije procesora poseduju viši stepen preklapanja, ali ne veći od 20. Nedostaci ovog rešenja su nizak stepen preklapanja i nedovoljno ubrzanje operacija koje imaju malo taktova (rad sa celim brojevima). Slično rešenje je i kod **vektorskih** računara, gde se operacije umesto nad pojedinačnim podatkom, izvršavaju nad nizom podataka.

Drugo rešenje je paralelizam višeg nivoa, koje karakteriše paralelni računarski sistem, sa više procesora. Više nivoe paralelizma karakteriše globalna paralelizacija celog programa, za razliku od paralelizma niskog nivoa, gde se paralelizuju procesorske operacije. Program se deli na delove, koji se mogu nezavisno izvršavati i svaki takav deo se dodeljuje jednom od procesora na izvršavanje. Svi procesori su u datom modelu ravnopravni i nezavisni. (detaljnije o aspektima paralelizma videti u [17] i [18]).

Istraživanja iz paralelnih računara se mogu podeliti na nekoliko oblasti:

(a) Projektovanje i razvoj paralelnih računara. (detaljnije u [17] [18])

(b) Paralelni programski jezici. (osim u ovom radu, videti [21] - [28])

(c) Razvoj paralelnih algoritama i teorijska ocena njihove efikasnosti (detaljnije u [2] [19]).

U ovom radu se razmatraju paralelni programski jezici. Funkcionalni programski jezici, kao što smo ranije videli, su pogodni za implementaciju na paralelnim računarima.

1.3.1. Modeli računara

Postoji više podela računarskih sistema. Jedna od najpoznatijih je Flynn-ova podela (videti [15], a može i [2] [17] [19]), pri čemu se računari po arhitekturi dele na:

SISD (Single Instruction Single Data)

Računari iz ove klase imaju jedan procesor koji prima jedan tok instrukcija i operiše nad jednim tokom podataka. Najveći broj današnjih računara spada u ovaj model koji je projektovao John von Neumann sa saradnicima krajem 40-tih. Algoritmi za ovaj model su sekvencijalni, jer postoji samo jedan procesor.

MISD (Multiple Instruction Single Data)

Kod ovog modela postoji N procesora, pri čemu svaki poseduje po jednu kontrolnu jedinicu, ali svi dele isti tok podataka. Pri svakom koraku, svi procesori obradjuju istovremeno jedan podatak prihvaćen iz (zajedničke) memorije. Svaki procesor obradjuje podatak prema instrukciji koju prima sa sopstvene upravljačke jedinice. Ovaj model je pogodan ukoliko nad istim podatkom treba izvršiti više operacija u isto vreme.

SIMD (Single Instruction Multiple Data)

Ovaj model karakteriše N identičnih procesora. Svaki procesor poseduje sopstvenu lokalnu memoriju. Svi procesori rade pod kontrolom jedne kontrolne jedinice, ili svaki procesor ima svoju kopiju istog programa u lokalnoj memoriji. Izvršavanje je sinhrono, odnosno, svaki procesor u isto vreme izvršava istu instrukciju, i ima svoj tok podataka.

MIMD (Multiple Instruction Multiple Data)

U ovom modelu svaki procesor poseduje kontrolnu jedinicu, aritmetičko logičku jedinicu i lokalnu memoriju. Svaki procesor ima svoj tok instrukcija i podataka, tako da procesori mogu izvršavati različite programe nad različitim podacima, što znači da tipično rade asinhrono. Iako je najmoćniji model, sa višeprocesorskim računarima ovog modela je najteže raditi, jer postoje problemi koji se ne javljaju kod ostalih modela:

❶ Dodela procesa procesoru,

❷ Čekanje procesa na slobodan procesor,

❸ Razmena podataka izmedju više procesora ili procesa

1.3.2. Komunikacija izmedju procesora

SIMD i MIMD modeli računara se, po načinu komunikacije, mogu podeliti na računare sa komunikacijom preko zajedničke memorije (Shared Memory), i komunikacijom prosledjivanjem poruka (Message Passing).

1.3.2.1. Komunikacija prosledjivanjem poruka

Svaki procesor poseduje vlastitu memoriju, a medjuprocesorska komunikacija se odvija preko veza izmedju njih. Načini povezivanja ([2] [17] [19]) :

(a) Svaki sa svakim

(b) Jednodimenzioni niz

(c) Pravougaona mreža

(d) Hiperkocka

(e) Drvo

(f) Prsten

(g) Dvostruki prsten

. . . . .

1.3.2.2. Uporedna analiza

Paralelni računarski sistem sa komunikacijom preko zajedničke memorije, iako teorijski moćniji i fleksibilniji, nije u praksi dao rezultate koji se u teoriji predvidjaju. Razlozi su u težoj implementaciji dela za paralelno čitanje odnosno upis podataka i u njegovom sporijem izvršavanju, ukoliko je broj procesora veliki (videti [2]). Programiranje na takvim sistemima je olakšano, jer operativni sistem sam vodi računa o paralelnom čitanju i upisu. Sistemi sa prosledjivanjem poruka su lakši za projektovanje i implementaciju, ali je na njima teže programirati, jer sam programer mora da vodi računa o sinhronizaciji procesora i razmeni podataka izmedju njih. Kod sistema sa zajedničkom memorijom, operativni sistem obezbedjivanjem paralelnog čitanja, odnosno upisa podataka, rešava sam takve probleme.

U programiranju višeprocesorskih računara cilj je minimizacija vremena izvršavanja. Preduslov za to je maksimalna iskorišćenost procesora (da pojedini procesori čekaju što je moguće manje).

Medjuprocesorska komunikacija, kod sistema sa prosledjivanjem poruka, takodje, bitno utiče na vreme izvršavanja. Vreme potrebno za prenos odredjenog

podatka, izmedju različitih procesora, je ponekad, nekoliko puta veće od vremena potrebnog za aritmetičku operaciju nad njim (videti Dodatak A.1.).

Medjutim, povećanje iskorišćenosti procesora zahteva i veću komunikaciju, pa se moraju odrediti takve vrednosti (komunikacije izmedju procesora i iskorišćenosti procesora) koje obezbedjuju minimalno vreme izvršavnja.

Te vrednosti zavise direktno od prirode problema, i to od toga da li se i kako problem može razložiti na potprobleme, koji ne zavise jedni od drugih. Tada se svi potproblemi mogu istovremeno izvršavati na različitim procesorima.

Programiranje višeprocesorskih računara je višestruko složenije od programiranja jednoprocesorskih računara. Mnogo je više faktora koji bitno utiču na ukupno vreme izvršavanja (detaljnije videti u [2] [18] [19]). I analiza efikasnosti algoritama i vremena izvršavanja programa na višeprocesorskim računarima je mnogo složenija u odnosu na sekvencijalne algoritme i programe.

1.4. Karakteristike paralelnog programiranja na transpjuterima

Vreme potrebno za prenos jednog podatka izmedju dva procesora je oko 4 puta duže u odnosu na aritmetičku operaciju na jednom procesoru, nad tim podatkom. (videti Dodatak A, deo: A1. Tehničke karakteristike). Intuitivno se nameće ideja da treba smanjiti komunikaciju. Naravno, što više smanjujemo komunikaciju, smanjuje se i razmena podataka, pa se smanjuje i iskorišćenost procesora. Prema tome treba odrediti kompromis izmedju smanjenja komunikacije i smanjenja iskorišćenosti procesora.

Problemi pogodni za paralelizaciju na transpjuterima su, uglavnom, oni sa velikim brojem računskih operacija, a malim brojem ulazno-izlaznih operacija. Problemi sa velikim brojem ulazno-izlaznih operacija, a malim brojem računskih operacija, nisu pogodni za paralelizaciju na transpjuterima. Razlog je mogućnost paralelizacije računskih operacija, uz nemogućnost paralelizacije ulazno-izlaznih operacija. Na primer, ulaz i izlaz podataka je direktno dopušten samo 1. transpjuteru, a ostalim transpjuterima su dozvoljene samo indirektne ulazno-izlazne operacije, slanjem poruke prvom transpjuteru, odnosno prijemom poruke od njega. Prvi transpjuter, kao posrednik, tada izvrši odgovarajuće ulazno-izlazne operacije.

S obzirom na vreme trajanja komunikacije, u slučaju velikog broja ulazno-izlaznih podataka, ulaz ili izlaz podataka može trajati mnogo duže od samog izračunavanja. Postoje neki problemi gde same ulazno izlazne operacije i odgovarajuće komunikacije dugo traju, pa u praksi izvršavanje na više procesora ne ubrzava izrvršavanje programa. Ponekad, zbog tehničkih ograničenja ili brzine komunikacije, algoritmi koji su teorijski jako efikasni, u praksi pokažu loše rezultate.

Primer je sabiranje N brojeva na transpjuterskom sistemu sa n procesora. Iako je vreme potrebno za izračunavanje $O(\frac{N}{n} \log n)$ (videti primer sabiranja N brojeva na binarnom drvetu u [2]), za ulaz podataka je potrebno vreme O(N), a za prosledjivanje sabiraka do svakog transpjutera je potrebno O(N logn) komunikacija. Dakle, vreme izvršavanja je O(N logn), što je više od O(N) koliko je potrebno da bi se izračunao zbir N brojeva na jednom procesoru!

Ako se koriste procesi, postoji problem deljenja resursa, pa rezultat rada nekoliko potpuno ispravnih programa, ako se izvršavaju paralelno, može dati nepredvidljive rezultate. To se dešava, na primer, ako više procesa pokušava da upisuje podatke u istu globalnu promenljivu, a ne koristi se zaštita upisa.

Kao što smo ranije naglasili, imajući u vidu data ograničenja, na transpjuterima se mogu uspešno paralelizovati, uglavnom, oni problemi gde nema mnogo ulaznih i izlaznih podataka, ali ima puno računanja sa njima. U većini takvih slučajeva se problem može uspešno podeliti na potprobleme, koji se mogu izvršavati nezavisno, odnosno paralelno. Naravno, to je samo neophodan, ali ne i dovoljan uslov da bi se problem mogao uspešno paralelizovati na transpjuterima.

2. Teorijska osnova

2.1. S-izrazi

Osnova programskog jezika LISP su S-izrazi (videti [1] [3] [4]).

Definicija 2.1.1.

Broj je niz cifara kome prethodi, opciono, znak minus.

Definicija 2.1.2.

Simbol je niz znakova koji nije broj, i ne sadrži znakove **( ) ;** Simbol predstavlja ime promenljive ili ime funkcije.

Definicija 2.1.3.

S-izraz je:

(a) Broj

(b) Simbol

(c) Lista **(S_1 S_2 . . . . S_n)** od nula ili više S-izraza

Definicija 2.1.4.

Nula-lista je lista sa nula članova **()**. Ona predstavlja i logičku konstantu **netačno**, a simbol T predstavlja logičku konstantu **tačno**.

Definicija 2.1.5.

Simbolička konstanta je S-izraz kome prethodi apostrof. Vrednost je ili broj ili simbolička konstanta.

Definicija 2.1.6.

U Backus-ovoj notaciji bi prethodne definicije glasile:

<cifra> ::= 0 | 1 | 2 | 3| 4 | 5 | 6 | 7 | 8 | 9

<slovo> ::= A | ...| Z | a | .....| z

<otvorena zagrada> ::= (

<zatvorena zagrad> ::=)

<znak> ::= <slovo> | <dozvoljeni specijalni znak>

<broj> ::= [-] <cifra> { <cifra>}

<simbol> ::= <znak> { <znak> | <cifra> }

<simbolička konstanta> ::= ' <S-izraz>

<vrednost> ::= <broj> | <simbolička konstanta>

<ime> ::= <simbol>

<ime promenljive> ::= <ime>

<ime funkcije> ::= <ime>

<izraz> ::= <vrednost> | <ime promenljive> | <otvorena zagrada> <ime funkcije> <lista izraza> <zatvorena zagrada>

<lista izraza> ::= <izraz> | {<izraz>}

<S-izraz> ::= <vrednost> | <simbol> | <otvorena zagrada> {<S-izraz>} <zatvorena zagrada>

2.2. Grafovi

Manji broj definicija i tvrdjenja u ovom odeljku su opšteg tipa, a više o tome se može naći u [16]. Preostale definicije i tvrdjenja, i svi dokazi u ovom odeljku, su rad samog autora.

2.2.1. Opšti deo

Definicija 2.2.1.1.

Strukturu **G=<N,A>** nazivamo **graf** ukoliko je **N** neprazan konačan skup, a $\mathbf{A} \subseteq \mathbf{N \times N}$. Elemente iz skupa **N** nazivamo čvorovima grafa, a elemente iz skupa **A** nazivamo ivicama grafa.

Definicija 2.2.1.2.

Graf **G=<N,A>** nazivamo **neorjentisanim** ukoliko važi:

(a) $(\forall x \in N)\ (x,x) \in A$

(b) $(\forall x \in N)\ (\forall y \in N)\ \ ((x,y) \in A \Rightarrow (y,x) \in A)$.

Broj ivica neorjentisanog grafa je broj različitih skupova **{ x,y }**, gde je $(x,y) \in A$ i $x \neq y$.

U suprotnom graf nazivamo **orjentisanim**. Broj ivica orjentisanog grafa je broj elemenata skupa **A**.

Definicija 2.2.1.3.

U orjentisanom grafu **G** kažemo da je čvor **y** susedan čvoru **x** ukoliko je $(x,y)\in A$.

Definicija 2.2.1.4.

Niz čvorova grafa $a_i\in N$, nazivamo **put**, ukoliko važi:

$((\forall i)\ (\forall j)\ (i\neq j \Rightarrow a_i\neq a_j))\ \wedge\ (\forall i)\ (1 \le i \le n-1)\ (x_i,x_{i+1})\in A.$

a_1 nazivamo početni, a a_n završni čvor datog puta.

Definicija 2.2.1.5.

Graf **G** je **cikličan,** odnosno, ima bar jedan ciklus ukoliko važi:

$(\exists x\in N)\ \ (\exists\ a_1, a_2, \ldots .,a_n)\qquad (\forall i)\ (\forall j)\ ((i\neq j \Rightarrow a_i\neq a_j)\wedge a_1=x\ \ \wedge\ \ a_n=x)\wedge$
$(\forall i)\ (1 \le i \le n-1)\ (x_i,x_{i+1})\in A.$

odnosno ako postoji put u kome su početni i završni čvor isti.

Definicija 2.2.1.6.

Za dati graf **G**, **stepen** čvora **x**, je broj njegovih susednih čvorova.

Definicija 2.2.1.7.

Ukoliko za dati graf **G,** svaki njegov čvor ima stepen jednak **s** ili jednak **0**, kažemo da je graf **G semiregularan** stepena **s**.

Ukoliko za dati graf **G,** svaki njegov čvor ima stepen jednak **s**, kažemo da je graf **G regularan** stepena **s**.

Definicija 2.2.1.8.

Za dati graf **G** definišimo **maksimalni stepen** grafa kao

max { stepen od **x | x**$\in$ **N}**

odnosno kao maksimum stepena svojih čvorova.

Definicija 2.2.1.9.

Orjentisan graf **G** je **orjentisano drvo** ukoliko:

$(\exists r\in N)\ \ (\forall x\in N)$ **(∃!** put u **G** iz **r** do **x)**

Čvor **r** zovemo koren drveta.

Definicija 2.2.1.10.

Semiregularno orjentisano drvo stepena 2 nazivamo **orjentisano binarno drvo**.

Tvrdjenje 2.2.1.11.

Koren orjentisanog drveta nije sused nijednog čvora.

Dokaz: Neka je r koren orjentisanog drveta. Pretpostavimo suprotno, neka r jeste sused nekog čvora x. Tada do x postoje bar 2 različita puta, jedan put je direktan od r do x, a drugi je r,....,x,r,....,x), što je u suprotnosti sa definicijom orjentisanog drveta. Kontradikcija!

Dakle, koren nije sused nijednog čvora.

Tvrdjenje 2.2.1.12.

Orjentisano drvo ima jedinstven koren.

Dokaz: Pretpostavimo suprotno, da orjentisano drvo ima bar 2 korena. Neka su $\mathbf{r_1}$ i $\mathbf{r_2}$ koreni drveta. Tada za neki čvor **x** postoji put iz $\mathbf{r_1}$, ali i put iz $\mathbf{r_1}$ do $\mathbf{r_2}$ pa iz $\mathbf{r_2}$ do $\mathbf{r_1}$, pa iz $\mathbf{r_1}$ do **x**, što je drugi (različiti) put od $\mathbf{r_1}$ do **x**. Kontradikcija!

Dakle, svako orjentisano drvo ima tačno 1 koren.

Tvrdjenje 2.2.1.13.

Orjentisano drvo nema ciklusa.

Dokaz: Pretpostavimo suprotno, da postoji ciklus (put) koji polazi iz nekog čvora **x** i završava takodje u njemu. Neka je **r** koren tog orjentisanog drveta. Tada od **r** do **x** postoji direktan put, ali i put od **r** do **x**, pa zatim put od **x** do **x** (ciklus). Postoje dva različita puta od **r** do **x**! Kontradikcija!

Dakle, orjentisano drvo nema ciklusa.

2.2.2. RS kompletni grafovi

Definicija 2.2.2.1.

Graf **G=<N,A>** nazivamo **RS kompletan** ukoliko je svaki čvor stepena najviše 3, i ukoliko se na skupu **N** mogu definisati relacije **R** i **S** takve da je:

(a) $(x,y)\in R\cup S \Rightarrow (x,y)\in A$

(b) $(\exists!\ r\in N)\ (\forall x\in N) \quad ((x,r)\notin R \wedge (x,r)\notin S)$,
odnosno postoji jedinstven čvor **r,** za koji ne postoji čvor koji je u relaciji **R** ili **S** sa njim. Taj čvor nazivamo **korenom.**

(c) $(\forall x\in N\setminus\{r\})\ (\exists!\ y\in N)\ \ (y,x)\in R\cup S$

odnosno za svaki čvor **x** osim korena **r**, postoji jedinstven čvor **y** kome je on susedan, i sa kojim je u relaciji **R** ili **S**.

(d) za svaki čvor **x** postoji najviše jedan čvor sa kojim je u relaciji **R**, i najviše jedan čvor sa kojim je u relaciji **S**.

(e) $R\cap S = \varnothing$, odnosno
ukoliko su dva čvora u relaciji **R** tada nisu u relaciji **S**, i obratno.

(f) Ne postoji RS ciklus, odnosno ne postoji niz $x_1, x_2, ..., x_n$ $(x_1 = x_n)$, takav da važi $(x_i, x_{i+1})\in R\cup S$.

Tvrdjenje 2.2.2.2.

Orjentisano binarno drvo je **RS kompletan** graf.

Dokaz: Svaki čvor orjentisanog binarnog drveta ima dva ili nijednog suseda. Ukoliko čvor **x** ima dva suseda **y** i **z**, definišimo $(x,y)\in R$, $(x,z)\in S$, $(x,z)\notin R$, $(x,y)\notin S$. Ako čvor nema suseda, nije u relaciji **R** ni **S**. Dakle $R\cup S = A$

(a) Pošto je $R\cup S = A$, sledi da je: $(x,y)\in R\cup S \Rightarrow (x,y)\in A$.

(b) Pošto koren orjentisanog drveta nije sused nijednog čvora, nijedan čvor nije u relaciji **R** ili **S** sa njim.

(c) Svaki čvor **x,** osim korena, je susedan nekom čvoru **y**, pa važi $(y,x)\in R$ ili $(y,x)\in S$.

(d) Sledi iz definicije relacija **R** i **S**.

(e) Takodje sledi iz definicije relacija.

(f) Pretpostavimo suprotno, odnosno neka postoji RS ciklus. Pošto je $R\cup S = A$, sledi da dato orjentisano binarno drvo ima ciklus, što je u suprotnosti sa tvrdjenjem 2.2.1.13. Dakle, ne postoji RS ciklus.

Definicija 2.2.2.3.

U RS kompletnom grafu niz $x_1, x_2, ..., x_n$, takav da je x_1 koren, i da važi $(\forall i)$ $(1 \leq i<n)$ $(x_i, x_{i+1}) \in R \cup S$, nazivamo **RS put**. Čvor x_n nazivamo završetak RS puta.

Tvrdjenje 2.2.2.4.

Za svaki čvor RS kompletnog grafa, koji nije koren, postoji jedinstven RS put čiji je on završetak.

Dokaz: Pretpostavimo suprotno, odnosno, ili postoje bar 2 različita RS puta, ili ne postoji nijedan.

Pretpostavimo da za proizvoljni čvor x postoje bar 2 RS puta čiji je on završetak. Neka su to RS putevi: $a_1, a_2, ..., a_n$ i $b_1, b_2, ..., b_m$. Tada je $a_n=b_m=x$. Pošto su to RS putevi važi $(a_{n-1}, x) \in R \cup S$ i $(b_{m-1}, x) \in R \cup S$, što protivreči delu c) definicije RS kompletnog grafa.

Pretpostavimo da za proizvoljni čvor $x=x_0$ ne postoji nijedan RS put čiji je on završetak. Neka je n broj čvorova grafa.

Pošto x nije koren, $(\exists!\ x_1)$ $(x_1, x) \in R \cup S$. Čvor x_1 ne može biti koren, jer je to u suprotnosti sa pretpostavkom da ne postoji RS put čiji je x završetak (postojao bi RS put x_1, x). Takodje ne postoji RS put čiji je x_1 završetak jer bi u suprotnom nadovezivanjem čvora x na dati put, postojao i RS put čiji je završetak x.

Dakle $(\exists!\ x_2)$ $(x_2, x) \in R \cup S$. Analogno prethodnom slučaju x_2 nije koren, niti završetak nekog RS puta. Takodje $x_2 \notin \{x_0, x_1\}$, jer bi u suprotnom postojao RS ciklus, što protivreči delu f) definicije RS kompletnog grafa.

Analogno postoje čvorovi $x_3, x_4, ..., x_{n+1}$ takvi da $(\forall i)$ $(1 \leq i \leq n)$ $(x_{i+1}, x_i) \in R \cup S$. Pošto graf ima ukupno n čvorova, $(\exists i,j)$ $(i<j)$ $x_i = x_j$, pa postoji RS ciklus $x_i, x_{i+1}, ..., x_j$, što protivreči delu f) definicije RS kompletnog grafa. Kontradikcija!

Dakle, postoji jedinstven RS put do datog čvora.

Tvrdjenje 2.2.2.5.

Orjentisano drvo, čiji je neki čvor stepena većeg od 2, **ne može biti** RS kompletan graf.

Dokaz: Pretpostavimo suprotno, da orjentisano drvo čiji je neki čvor stepena bar 3, može biti RS kompletan graf. Ukoliko je r koren takvog drveta, on mora biti i koren RS kompletnog grafa (jer $(\forall i)$ $(x_i,r) \notin A \Rightarrow (x_i,r) \notin R \cup S$, a koren RS kompletnog grafa je jedini takav čvor). Neka je **t** čvor čiji je stepen bar 3. Pošto t može biti u relaciji R odnosno S, sa po najviše jednim čvorom, postoji bar jedan od suseda sa kojim nije u relaciji (ni R ni S). Obeležimo taj čvor sa x. Pošto je dati graf RS kompletan, x mora biti završetak nekog RS puta $(r=a_1, a_2, ..., a_n=x)$. Iz $(a_i, a_{i+1}) \in R \cup S$ sledi $(a_i, a_{i+1}) \in A$, pa je to i put u orjentisanom drvetu. Dakle, do čvora x u datom orjenitasnom drvetu postoje najmanje dva puta (jedan je **r,....,t,x** ; a drugi je **$r=a_1, a_2, ..., a_n=x$)**, što protivreči definiciji orjentisanog drveta.

Dakle, orjentisano drvo čiji je neki čvor stepena većeg od 2, ne može biti RS kompletan graf.

2.2.3. Zaključak

Struktura veza između transpjutera može se, na očigledan način, predstaviti orjentisanim grafom. Tehnička ograničenja transpjutera (videti [12]), postavljaju dodatni zahtev da taj graf ne bude proizvoljan. Matematički model mogućeg povezivanja transpjutera predstavljen je upravo pojmom RS kompletnog grafa[1] .

S obzirom da svaki transpjuter ima svoju memoriju (i svoje stekove za argumente i za komunikaciju), jedan prirodan i jednostavan način komunikacije među procesorima bio bi sledeći: procesor koji prosleđuje funkciju za obradu drugom procesoru, stavlja argumente na stek za argumente drugog procesora. Kada drugi procesor završi sa obradom, on na svom steku za komunikaciju ostavlja rezultat obrade i javlja prvom procesoru da je slobodan. Prvi procesor tada može preuzeti rezultat.

Pri ovom načinu komunikacije treba voditi računa o sledećoj mogućnosti konflikta: ako bi dva različita procesora koristila isti (treći) procesor za obavljanje nekih svojih podzadataka, nebi bilo načina da se ustanovi koji procesor treba da preuzme koji rezultat. Prema tome, ovaj koncept komuniciranja, među procesorima, zahteva da svaki procesor prima zadatke samo od jednog procesora, što prirodno upućuje na strukturu veza između transpjutera u obliku drveta. Da bi to drvo bilo RS kompletan graf, ono prema tvrdjenju 2.2.2.5, ne sme imati čvorove stepena većeg od dva.

Iz navedenog se može zaključiti da je binarno drvo izbor koji se nameće iz dva razloga: da bi se dobio RS kompletan graf i da bi se postigla veoma jednostavna komunikacija između procesora.

[1] Dati pojam uvodi autor, kao matematičku formalizaciju tehničkih ograničenja transpjutera

3. Implementacija jezgra LISP-a na jednoprocesorskim računarima

Implementacija na PC računarima se sastoji iz nekoliko celina, od kojih svaka sadrži po više procedura. Osnovni koncept i deo materijala je preuzet iz [1], a drugi deo je rad samog autora, kao i neka poboljšanja. Neke ugradjene funkcije iz [1] se drugačije nazivaju, tj. nazivi su uskladjeni sa Common LISP-om. Odredjene funkcije nisu postojale pa su implementirane (**load, cond**).

3.1. Implementirane funkcije

Funkcije u LISP-u (ugradjene i korisničke) se pozivaju na sledeći način:

(imefunkc arg1 arg2 . . . .argn)

Ova verzija LISP-a sadrži sledeće ugradjene funkcije:

(defun imefunkc **(**arg1 arg2 . . . argn**) (** definicija-funkcije **))**

Definiše funkciju **imefunkc** čiji su formalni argumenti **arg1 . . . argn** i čija se definicija nalazi u **definicija-funkcije**.

(load "imedatoteke"**)**

Učitava datoteku **imedatoteke** sa diska, koristi je kao ulaznu datoteku, za učitavanje definicija funkcija ili poziva funkcija. koji se nalaze u datoteci. Po završetku obrade datoteke, kontrola ulaza se predaje korisniku (ulaz se vrši sa tastature). Funkciju **load** je moguće primeniti proizvoljan broj puta u toku rada, kao i mogućnost da se unutar pozvane datoteke nalazi još neka **load** funkcija.

quit

Završetak programa. Posle ove komande (funkcije), prekida se rad interpretera, i kontrola se predaje operativnom sistemu.

(if uslov funkc1 funkc2**)**

Ispituje vrednost S-izraza **uslov**, i ako je uslov tačan (vrednost mu je različita od nula S-izraza) primenjuje se funkcija **funkc1**, a u suprotnom **funkc2**.

(cond (uslov1 funkc1) (uslov2 funkc2) . . . . (uslovn funkcn)**)**

Ispituje vrednost S-izraza **uslov1** i ukoliko je tačan izvršava funkciju **funkc1** i time završava rad funkcije **cond**. U suprotnom dalje ispituje **uslov2** i ukoliko je on tačan izvršava funkciju **funkc2**, i time završava rad funkcije **cond**. Ako nijedan od prethodnih uslova nije bio tačan, ispituje dalje uslove **uslov3, ..., uslovn,** i izvršava odgovarajuću funkciju.

(while uslov funkc**)**

Ispituje vrednost S-izraza **uslov** i izvršava funkciju **funkc** , sve dok je vrednost S-izraza **uslov** tačna. Kada **uslov** postane netačan (nula S-izraz), **while** funkcija prekida rad. Ukoliko je vrednost **uslov** na početku bila netačna, funkcija **funkc** se ne izvršava nijednom, već se odmah prelazi na sledeću funkciju.

(setq prom vred**)**

Promenljivoj **prom** dodeljuje vrednost **vred**. Vrednost može biti bilo koji S-izraz.

(begin izraz1 izraz2 . . . . . izrazn**)**

Izračunava redom vrednosti izraza **izraz1, izraz2 , ..., izrazn**.

(+ arg1 arg2**)**

Nalazi vrednost zbira **arg1** i **arg2.**

(- arg1 arg2**)**

Nalazi vrednost razlike **arg1** i **arg2.**

(* arg1 arg2**)**

Nalazi vrednost proizvoda **arg1** i **arg2.**

(/ arg1 arg2**)**

Nalazi celobrojni deo količnika **arg1** i **arg2**.

(= arg1 arg2**)**

Ispituje da li su vrednosti S-izraza **arg1** i **arg2** jednake ili ne. Ukoliko su jednaki vraća tačnu vrednost, a inače vraća netačnu vrednost (nula S-izraz).

(< arg1 arg2**)**

Ispituje da li je brojna vrednost S-izraza **arg1** manja od **arg2** ili ne. Ukoliko je odgovor potvrdan vraća tačnu vrednost, a inače vraća netačnu vrednost (nula S-izraz).

(> arg1 arg2**)**

Ispituje da li je brojna vrednost S-izraza **arg1** veća od **arg2** ili ne. U slučaju da jeste, vraća tačnu vrednost, a inače vraća netačnu vrednost (nula S-izraz).

(cons izr1 izr2**)**

Konstruiše listu čiji je početak (**poc** odnosno **head**) S-izraz **izr1**, a ostatak (**ost** odnosno **tail**) lista **izr2**. Ova funkcija je inverzna u odnosu na funkcije **car** i **cdr,** odnosno **(car (cons izr1 izr2))** je **izr1**, a **(cdr (cons izr1 izr2))** je **izr2.** Isto tako je **(cons (car lista) (cdr lista))** jednako **lista**.

(car izraz**)**

Vraća vrednost početka (head) S-izraza **izraz**.

(cdr izraz**)**

Vraća vrednost nastavka (tail) S-izraza **izraz**.

(number? izraz**)**

Ispituje tip S-izraza **izraz**, i vraća tačnu vrednost ako je to broj, a inače, ako je to simbol, lista, ili S-izraz nekog drugog tipa, vraća netačnu vrednost (nulti S-izraz).

(symbol? izraz**)**

Ispituje tip S-izraza **izraz**, i vraća tačnu vrednost ako je to simbol, a inače, ako je to broj, lista, ili S-izraz nekog drugog tipa, vraća netačnu vrednost (nulti S-izraz).

(list? izraz**)**

Ispituje tip S-izraza **izraz**, i vraća tačnu vrednost ako je to lista, a inače, ako je to S-izraz nekog drugog tipa, vraća netačnu vrednost (nulti S-izraz).

(null? izraz**)**

Ispituje tip S-izraza **izraz**, i vraća tačnu vrednost ako je to nulti S-izraz, inače, ako je to broj, simbol, lista, ili S-izraz nekog drugog tipa, vraća netačnu vrednost (nulti S-izraz). Ova funkcija predstavlja i logičku negaciju.

(print izraz**)**

Štampa vrednost S-izraza **izraz**. Moguće je u nekim slučajevima štampanje vrednosti i bez navodjenja službene reči **print**, navodjenjem samo S-izraza.

(T)

Oznaka za tačan iskaz (S-izraz).

()

Nulti S-izraz. Takodje i oznaka za netačni iskaz.

3.2. Način implementacije

Program se sastoji od celina:

❶ Definicije

❷ Deklaracije promenljivih

❸ Upravljanje memorijom

❹ Podrška osnovnim tipovima podataka

❺ Operacije sa imenima funkcija (ugradjenim i korisničkim)

❻ Ulaz podataka i definicija korisničkih funkcija

❼ Postavljanje parametara

❽ Manipulacija S-izrazima

❾ Izračunavanje vrednosti S-izraza

❿ Glavni program

U ovom odeljku će biti dat samo opšti koncept implementacije i najvažnijih procedura. Detaljni opis implementacije i svih procedura sadrži Dodatak B (odeljak B.1 Opis implementacije na jednoprocesorskim računarima).

3.2.1. Definicije

Ova celina sadrži definicije svih konstanti i tipova. Najznačajniji tip je **sizr,** koji je pokazivač na strukturu **sizrstr.** Data struktura predstavlja S-izraz.

3.2.2. Deklaracije promenljivih

U ovoj celini su date deklaracije promenljivih u programu. Najznačajnije su:

❶ Niz **argstek** je stek za smeštanje argumenata funkcija,

❷ Promenljiva **svedeffun** je pokazivač na listu koja sadrži sve definicije korisničkih funkcija.

3.2.3. Upravljanje memorijom

Procedure iz ove celine su namenjene za upravljanje memorijom. Memorija za S-izraze se, po potrebi, dodeljuje iz liste slobodnih S-izraza (**slobsizr** je pokazivač na nju), a po završetku korišćenja se automatski vraća (**garbage collection**). Lista

slobodnih S-izraza se alocira na početku rada interpretera i konstantne je veličine (zadata konstantom **velmem**). Ukoliko dodje do prekoračenja, odnosno do popunjavanja cele liste, sistem za upravljanje memorijom javlja odgovarajuću poruku. Upravljanje memorijom se vrši po metodu brojanja referenci (**reference counting)**.

Svaki S-izraz sadrži dodatno polje u kome se beleži broj referenci na njega (broj pokazivača koji pokazuju na datu stukturu). Pri prvom korišćenju se taj broj postavlja na 1 i povećava pri svakom novom korišćenju datog S-izraza. Po prestanku nekog korišćenja datog S-izraza broj se smanjuje. Ukoliko broj referenci za taj S-izraz postane 0, taj S-izraz nije u upotrebi, pa se ponovo smešta u listu slobodnih S-izraza.

Postoji više metoda za upravljanje memorijom (videti [1]). Nedostatak ovog metoda (u odnosu na neke druge metode) je u većem zauzeću memorije (brojač referenci u svakom S-izrazu). Medjutim, ovaj metod ima nekoliko prednosti u odnosu na ostale modele:

❶ Brzina izvršavanja

❷ Jednostavnost implementacije

❸ Nema problema sa defragmentacijom memorije

3.2.4. Podrška osnovnim tipovima podataka

Ova celina sadrži procedure za podršku osnovnim tipovima podataka. Date su procedure za alokaciju: izraza, liste izraza, liste imena, liste vrednosti, i stanja. Takodje su date procedure za nalaženje dužine liste imena i dužine liste vrednosti.

Alokacija liste imena se najviše koristi kod definisanja formalnih argumenata, a alokacija liste vrednosti kod stvarnih argumenata korisničkih funkcija.

Procedura za alokaciju strukture **izrazstr** se najčešće koristi za smeštanje definicije neke korisničke funkcije.

Pokazivač na strukturu koja sadrži listu formalnih argumenata i listu vrednosti tih argumenata je **stanje**.

3.2.5. Operacije nad imenima funkcija

Data celina sadrži procedure za manipulaciju imenima funkcija.

Na početku rada interpretera procedura **postaviim** postavlja imena svih ugradjenih funkcija. Procedure iz ove celine koriste se još i za:

❶ Dodavanje imena nove funkcije u listu imena, pri definisanju nove korisničke funkcije.

❷ Dopunu liste definicija svih funkcija, novom korisničkom funkcijom

❸ Za zadato ime funkcije, nalazi njenu definiciju, pri izračunavanju vrednosti korisničke funkcije.

❹ Štampa ime funkcije, čija se vrednost izračunava.

❺ Za dato ime ugradjene funkcije, nalazi odgovarajuću ugradjenu funkciju.

3.2.6. Ulaz podataka i definicija korisničkih funkcija

Procedure u ovoj celini koriste se za:

❶ Obradu ulaznih podataka, i njihov smeštaj u niz **ulazpod.**

❷ Iz niza **ulazpod** izbacuje sve vodeće praznine.

❸ Izdvajaju odgovarajuće strukture podataka iz ulaznog niza **ulazpod.** Vraćaju pokazivač na odgovarajuću strukturu podataka:

❹ Ispituje da li je neka vrednost brojnog, ili nekog drugog tipa.

3.2.7. Postavljanje parametara

Procedure u ovoj celini služe za postavljanje parametara. Najvažniji zadaci su im:

❶ Dodela vrednosti promenljivoj.

❷ Ispitivanje da li je dato ime promenljiva (ili funkcija)?

❸ Dobijanje trenutne vrednosti promenljive.

❹ Postavljanje praznog stanja.

❺ Nalaženje promenljive u datom stanju.

3.2.8. Manipulacija S-izrazima

U tekućoj celini su procedure za manipulaciju S-izrazima i nalaženje rezultata nekih operacija. Te procedure koriste se za:

❶ Štampanje S-izraza.

❷ Primenu aritmetičke ili logičke operacije (funkcije).

❸ Primenu date operacije

❹ Ispitivanje da li je tačna vrednost nekog izraza

❺ Nalazi broj argumenata operacije

3.2.9. Izračunavanje vrednosti izraza

Ovo je najvažnija celina u kojoj procedure služe za:

❶ Za izračunavanje vrednosti izraza (procedura **izracunaj**). Dati izraz može biti brojna ili simbolička konstanta, promenljiva, lista izraza ili funkcija.

❷ Za izračunavanje vrednosti liste izraza (procedura **izraclistu**).

Ukoliko je tekući izraz brojna ili simbolička konstanta, izračunavanje vrednosti je kopiranje te konstante.

Ako je izraz bio promenljiva, u zavisnosti od vrednosti polja **ofset** vrednost čitamo sa steka za čuvanje argumenata, ili iz liste **ukstanje** u kojoj su memorisana imena i vrednosti svih promenljivih.

U slučaju da je bila u pitanju lista izraza, izračunavamo svaki izraz iz date liste izraza.

Ukoliko je izraz predstavljao funkciju, mogući su sledeći slučajevi:

① Funkcija je korisnička, i tada prvo izračunamo vrednosti njenih argumenata, a zatim .primenimo datu korisničku funkciju (**primkorop**).

② U pitanju je kontrolna operacija (**if, while, cond,** ili **set).** Tada primenjujemo datu kontrolnu operaciju na njene argumente (**primkonop**).

③ U slučaju operacije koja vraća vrednost (+,-,*, /, ...), takodje prvo izračunamo vrednosti argumenata, a zatim je primenimo (**primvrop**).

S obzirom da funkcija može imati proizvoljan broj argumenata, izračunavanje vrednosti liste izraza je potrebno pri nalaženju vrednosti argumenata funkcije, .

3.2.10. Glavni program

Ova celina sadrži procedure, koje objedinjuju rad svih procedura u programu:

❶ Postavljaju se početne vrednosti.

❷ Poziva se procedura **ucitaj()** za učitavanje ulaznih podataka.

❸ Ispituje se da li je nastupio kraj rada interpretera (**"quit"** kao ulazni podatak).

❹ Ako se na ulazu nalazi definicija korisničke funkcije (**"defun"**),.izdvaja je. Pamti je zatim u listi korisničkih funkcija i štampa ime date funkcije.

❹ Ukoliko je u pitanju poziv funkcije, poziva funkciju **izracunaj** i izračunava vrednost date funkcije. Po završetku izračunavanja štampa vrednost funkcije. Ukoliko je u pitanju promenljiva, štampa njenu vrednost.

❺ Vraća se na korak ❷.

4. Implementacija LISP-a na transpjuterima

4.1. Metod implementacije

U implementaciji LISP-a na jednoprocesorskim mašinama (PC računarima) osnovni deo koji se može paralelizovati je deo za izračunavanje vrednosti izraza. S obzirom da je ulaz podataka dozvoljen samo 1. transpjuteru, jedino na njemu je moguće izvršavanje procedura za ulaz i izlaz. Procedure za manipulaciju funkcijama je praktičnije izvršiti na 1. transpjuteru, imajući u vidu relativno sporu komunikaciju, a ostalim transpjuterima poslati definicije funkcija koje su im potrebne pri evaluaciji.

Tehnička ograničenja transpjutera su:

❶ Svaki transpjuter može biti povezan sa najviše 4 druga transpjutera

❷ Svaki transpjuter mora biti resetovan od nekog transpjutera, preko jedne od svoje 4 veze. Jedino prvi transpjuter je resetovan od računara domaćina (hosta).

❸ Bilo koji transpjuter može resetovati najviše dva transpjutera, jedan preko sistemskog (R), a drugi preko podsistemskog (S) reset kanala.

Ova tehnička ograničenja se precizno u teoriji grafova (videti odeljak 2.2.2) opisuju pomoću RS kompletnog grafa maksimalnog stepena 4.

Iz tvrdjenja 2.2.2.2 sledi da arhitektura u obliku binarnog drveta, zadovoljava tehničke uslove (RS kompletan graf maksimalnog stepena 4), pa se može napraviti transpjuterski sistem te arhitekture. Ona se i koristi u ovom radu.

Transpjutere možemo podeliti u 3 grupe:

❶ Prvi transpjuter

❷ Transpjuteri, osim prvog, koji imaju "naslednike" (u ovoj implementaciji su to transpjuteri sa rednim brojevima 2. - 8.)

❸ Transpjuteri koji nemaju "naslednike" (ostalih 9 transpjutera)

Arhitektura (konfiguracija) transpjuterskog sistema se zadaje u datoteci sa nastavkom NIF. Sledi primer NIF datoteke za datu implementaciju LISP-a, gde se transpjuterski sistem sastoji od 17 transpjutera T800™:

```
1,    lisptr1,  R0,  0  ,  2  ,  3  ,    ;
2,    lisptr2,  R1,  4  ,  1  ,  5  ,    ;
3,    lisptr2,  S1,  6  ,  7  ,  1  ,    ;
4,    lisptr2,  R2,  2  ,  8  ,  9  ,    ;
5,    lisptr2,  S2,  10 ,  11 ,  2  ,    ;
6,    lisptr2,  R3,  3  ,  12 ,  13 ,    ;
7,    lisptr2,  S3,  14 ,  3  ,  17 ,    ;
8,    lisptr2,  R4,  18 ,  4  ,  19 ,    ;
9,    lisptr2,  S4,     ,     ,  4  ,    ;
10,   lisptr2,  R5,  5  ,     ,     ,    ;
11,   lisptr2,  S5,     ,  5  ,     ,    ;
12,   lisptr2,  R6,     ,  6  ,     ,    ;
13,   lisptr2,  S6,     ,     ,  6  ,    ;
14,   lisptr2,  R7,  7  ,     ,     ,    ;
17,   lisptr2,  S7,     ,     ,  7  ,    ;
18,   lisptr2,  R8,  8  ,     ,     ,    ;
19,   lisptr2,  S8,     ,     ,  8  ,    ;
```

Svaki red se sastoji od:

❶ Rednog broja transpjutera (1. mora biti povezan vezom 0 sa PC-em koji ima redni broj 0).

❷ Ime programa koji se izvršava na tom transpjuteru,

❸ R ili S a zatim redni broj transpjutera koji ga resetuje,

❹ Sa kojim transpjuterom je povezan vezom 0,

❺ vezom 1,

❻ vezom 2

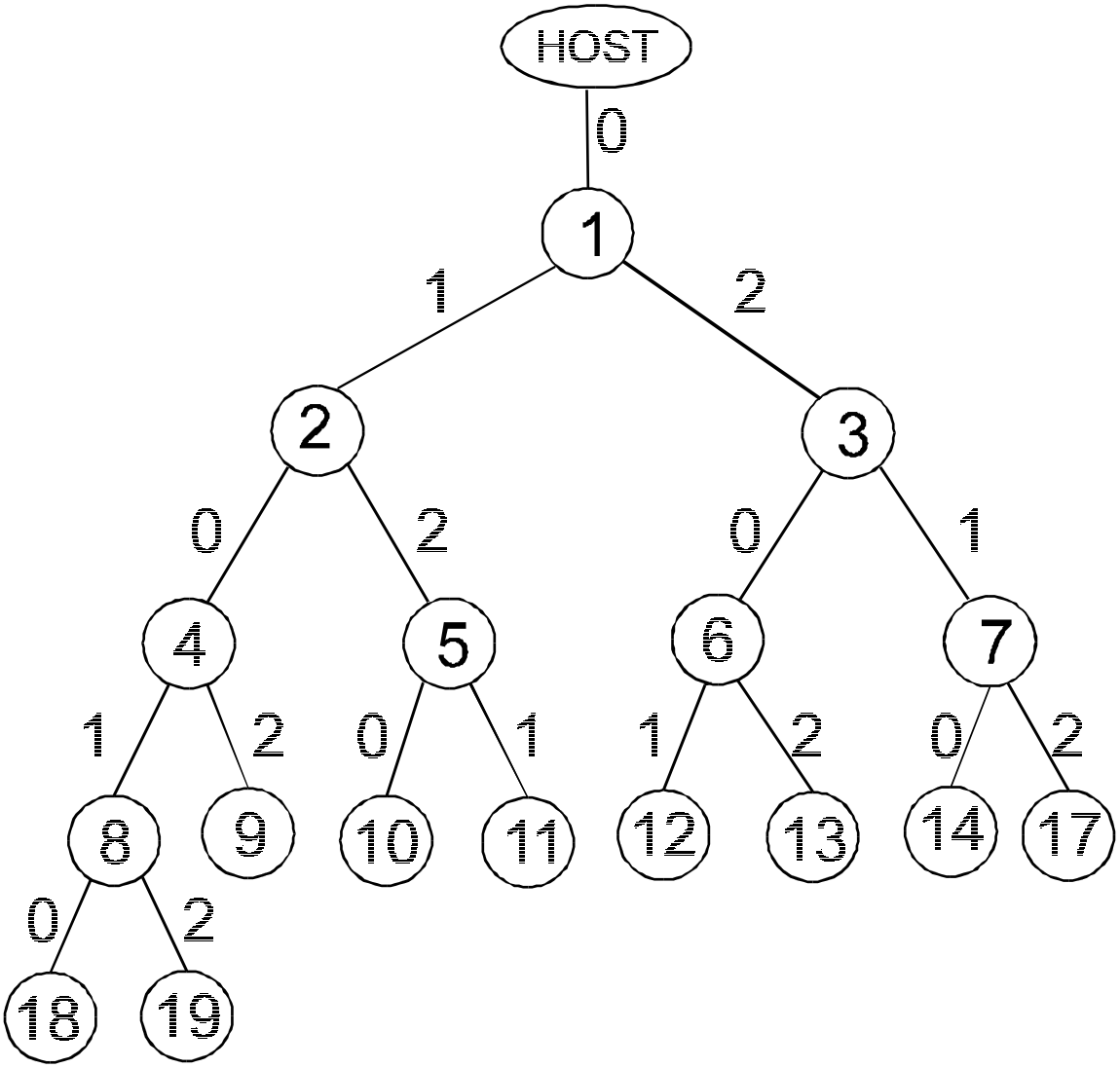
❼ vezom 3.

Ukoliko transpjuter nije povezan nekom vezom, dato mesto ostaje prazno.

Primer: Recimo za 5. transpjuter (5. red u datoteci) imamo podatke:

Redni broj je 5, na njemu se izvršava program LISPTR2, resetuje ga preko podsistemskog reset kanala 2. transpjuter (oznaka S2). Vezom 0 je povezan sa 10. transpjuterom, vezom 1 sa 11 transpjuterom, a vezom 2 sa 2 transpjuterom (koji ga kao što smo videli resetuje). Veza 3 je slobodna.

Konfiguracija transpjuterskog sistema zadata u NIF datoteci iz gornjeg primera predstavlja binarno stablo kao na slici 1. Detaljnije o konfigurisanju mreže transpjutera može se videti u [12].



slika 1.

4.1.1. Zadaci prvog transpjutera

Prvi transpjuter izvršava sledeće zadatke:

- ❶ Postavlja početne vrednosti.
- ❷ Učitava ulazne podatke.
- ❸ Obradjuje ulazne podatke i iz ulaznog niza izdvaja definicije funkcija.
- ❹ Date definicije funkcija smešta u memoriju (u strukturi **svedeffun).**
- ❺ Imena promenljivih i funkcija, takodje, memoriše (u niz **nizimena).**

❻ Izdvaja pozive funkcija iz ulaznog niza.

❼ Odlučuje da li je svrsishodnije da funkciju sam izvrši ili da je na izvršavanje pošalje nekom od svojih naslednika.

❽ Štampa izlazne rezultate.

Ukoliko je potrebno naći vrednost neke funkcije, ispituje da li je u pitanju ugradjena ili korisnička funkcija.

4.1.1.1. Poziv ugradjene funkcije

Ako je reč o pozivu ugradjene funkcije, prvi transpjuter vrši izračunavanje, jer je u većini slučajeva izračunavanje ugradjenih funkcija kratko. Pošto je reč o kratkom računanju, a imajući u vidu vreme potrebno za komunikciju izmedju transpjutera, necelishodno bi bilo slati takvu funkciju na računanje ostalim transpjuterima.

4.1.1.2. Poziv korisničke funkcije

Ako treba izračunati vrednost korisničke funkcije, program postupa na sledeći način:

❶ Ukoliko data korisnička funkcija sadrži samo pozive ugradjenih funkcija, prvi transpjuter sam vrši sva izračunavanja jer su u većini slučajeva ona kratka.

❷ U slučaju da pomenuta korisnička funkcija sadrži i pozive drugih korisničkih funkcija (može se i rekurzivno pozivati), ima smisla očekivati računanje većeg obima. Tada, ukoliko su slobodni neki "naslednici", pozivi tih korisničkih funkcija (koje služe pri računanju polazne korisničke funkcije) se šalju na izračunavanje "naslednicima". Ako nijedan od "naslednika" u tom trenutku nije bio slobodan, prvi transpjuter sam vrši računanje vrednosti datih funkcija.

4.1.2. Zadaci transpjutera koji imaju "naslednike"

Svaki od preostalih transpjutera, koji imaju "naslednike", (u slučaju 17 transpjutera to su oni sa rednim brojevima od 2. do 8.), čeka sve dok od "roditeljskog" transpjutera ne dobije komandu za računanje.

U trenutku kada dobije komandu za računanje, prima sa odgovarajućeg ulaznog kanala, od "roditeljskog" transpjutera sledeće podatke:

❶ Izraz (funkciju) koji treba da izračuna.

❷ Imena i vrednosti promenljivih potrebnih za računanje.

❸ Definicije funkcija koje su mu potrebne za računanje.

❹ Trenutni sadržaj steka za smeštanje argumenata

Dalje izračunava vrednost date funkcije na isti način kao i prvi transpjuter.

Ako nijedan od "naslednika" trenutno nije bio slobodan, dati transpjuter sam vrši računanje funkcije.

Kada dati transpjuter završi celokupno svoje računanje, šalje poruku prethodniku da je slobodan, a rezultat smešta na svoj stek za komunikaciju.

U trenutku u kome je "roditeljskom" transpjuteru potreban rezultat računanja koje je poslao na dati transpjuter, šalje poruku datom transpjuteru ("nasledniku"). Zatim dati transpjuter uzima potrebnu vrednost sa steka za komunikaciju, i šalje je "roditeljskom" transpjuteru.

4.1.3. Zadaci transpjutera koji nema "naslednike"

Svaki od transpjutera koji nema "naslednike" (u slučaju konfiguracije od 17 transpjutera, reč je o preostalih 9 transpjutera), čeka sve dok od "roditeljskog" transpjutera ne dobije komandu za računanje.

Kada dobije komandu za računanje, prima sa ulaznog kanala sve potrebne podatke (koji su isti kao i u slučaju transpjutera koji ima "naslednike").

Zatim dati transpjuter vrši celokupno izračunavanje vrednosti funkcije koja mu je poverena. Po tome se jedino razlikuje od transpjutera koji imaju "naslednike".

Dalji postupak je isti kao i kod transpjutera koji imaju "naslednike", odnosno šalje poruku "roditeljskom transpjuteru da je slobodan, i rezultat smešta na svoj stek za komunikaciju. Kada "roditeljski" transpjuter zatraži vrednost, dati transpjuter je uzima sa steka za komunikaciju i šalje "roditeljskom" transpjuteru.

4.2. Način implementacije

Kao osnova za implementaciju na transpjuterima iskorišćena je implementacija na jednoprocesorskim (PC) računarima. Prethodne celine su uglavnom ostale iste, uz poneku sitniju izmenu, a dodate su dve nove celine:

❶ Prenos argumenata

❷ Kontrolni deo.

Postoje dva dela programa:

① Prvi deo se izvršava na prvom transpjuteru i sadrži sve što sadrži i verzija za jednoprocesorske računare uz odgovarajuće dodatke.

② Drugi deo se izvršava na svim ostalim transpjuterima i osiromašen je za one procedure koje se ne mogu izvršavati na ostalim transpjuterima (ulazno-izlazne operacije i ostale procedure sličnog tipa).

4.2.1. Deo programa koji se izvršava na prvom transpjuteru

Promene u delu programa koji se izvršava na prvom transpjuteru, u odnosu na implementaciju na jednoprocesorskim računarima su:

❶ U celinama **definicije** i **deklaracije promenljivih** su dodate neke definicije i deklaracije promenljivih, koje su potrebne kasnije.

❷ Dodate su nove celine **prenos argumenata** i **kontrolni deo**, koje su specifične za implementaciju na transpjuterima.

4.2.1.1. Prenos argumenata

Paralelni C sadrži samo procedure za prenos celih brojeva ili znakova preko kanala. S obzirom da se javljaju potrebe prenosa (slanja i prijema) argumenata tipa **izraz, sizr, stanje** i **deffun,** koji su neophodni za izračunavanje vrednosti funkcije na transpjuteru "nasledniku", potrebne su i odgovarajuće procedure. Ova celina sadrži procedure koje vrše prenos datih kompletnih struktura podataka (i njihovih podstruktura).

4.2.1.2. Kontrolni deo

Ovo je najbitnija celina za paralelno izvršavanje. Vrši sledeće:

❶ Prima poruke sa ulaznih kanala i izvršava odgovarajuće akcije.

❷ Beleži transpjutere koji su završili prethodno izračunavanje i postali slobodni.

❸ Pri izračunavanju vrednosti funkcije, ispituje da li dati transpjuter ima naslednika, da li medju njima postoji neki slobodan i da li je dati izraz korisnička funkcija? Ukoliko su svi uslovi zadovoljeni, šalje datu funkciju na izračunavanje prvom slobodnom svom "nasledniku", inače joj sam izračunava vrednost.

❹ Šalje zahtev za izračunatom vrednošću, a zatim čeka, sve dok ne dobije datu vrednost.

4.2.2. Deo programa koji se izvršava na ostalim transpjuterima

Na ostalim transpjuterima su izbačene neke procedure kao nepotrebne, a uvedeno nekoliko novih.

U celini Prenos argumenata nove su procedure za rad sa stekom za komunikaciju (koje nisu potrebne na prvom transpjuteru).

U Kontrolnom delu postoji nekoliko dodatnih slučajeva:

❶ Prijem funkcije za izračunavanje (i svih potrebnih pripadajućih podataka) od "roditeljskog" transpjutera

❷ Zahtev "roditeljskog" transpjutera za izračunatom vrednošću.

❸ Prijem poruke o prestanku rada interpretera. Takvu poruku takodje šalje "roditeljski" transpjuter.

4.3. Efikasnost implementacije

Ova implementacija je efikasnija ukoliko se radi o većem računanju, ali i tu zavisi od strukture problema. Kod nekih problema je to bolje, a kod nekih je ubrzanje u odnosu na sekvencijalni algoritam malo. Problemi gde ima malo računanja se slabo ubrzavaju, ali je tu vreme izvršavanja malo, pa to nije mnogo ni bitno.

Testiranje je obavljeno sa nekoliko testova. S obzirom na činjenicu da je prenos informacija oko 4 puta duži u odnosu na izračunavanje, prihvatljivi rezultati se mogu dobiti samo ukoliko je mali broj ulaznih, odnosno izlaznih podataka, a veliki broj aritmetičkih operacija.

U tabelama 1. - 6. sva vremena izvršavanja su data u milisekundama. Greška u merenju vremena je najviše ±5 ms.

U različitim vrstama su dati rezultati rada za različite ulazne podatke.

U svakom redu su dati:

❶ Argument funkcije (to je u datim test-primerima, približno, dubina rekurzije).

❷ Rezultat funkcije (to je u datim test-primerima, približno, broj poziva funkcije, za poslednji nivo).

❸ Vreme izvršavanja na 1 transpjuteru

❹ Vreme izvršavanja na jednom PC računaru (386DX - 40 Mhz).

❺ Vreme izvršavanja na konfiguraciji od 3 transpjutera, i odgovarajuće ubrzanje, u odnosu na vreme izvršavanja na jednom transpjuteru.

❻ Vreme izvršavanja na konfiguraciji od 7 transpjutera, i odgovarajuće ubrzanje, u odnosu na vreme izvršavanja na jednom transpjuteru.

❼ Vreme izvršavanja na konfiguraciji od 15 transpjutera, i odgovarajuće ubrzanje, u odnosu na vreme izvršavanja na jednom transpjuteru.

❽ Vreme izvršavanja na konfiguraciji od svih 17 transpjutera, i odgovarajuće ubrzanje, u odnosu na vreme izvršavanja na jednom transpjuteru.

Primer 1: Funkcija koja 2 puta rekurzivno poziva samu sebe:

```
(defun t2 ( x )
    (if (= x 0)
        1
        ( + (t2 (- x 1)) (t2 (- x 1)))))
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
10	1024	741	385	428	1.731	232	3.194	125	5.928	125	5.928
11	2048	1481	714	850	1.742	457	3.241	237	6.249	237	6.249
12	4096	2961	1483	1695	1.747	906	3.268	461	6.423	461	6.423
13	8192	5921	2856	3384	1.75	1804	3.282	911	6.499	910	6.507
14	16384	11841	5713	6764	1.751	3541	3.344	1809	6.546	1808	6.549
15	32768	23681	11481	13522	1.751	7073	3.348	3605	6.569	3604	6.57
16	65536	47362	22906	27039	1.752	14138	3.35	7197	6.581	7197	6.581
17	131072	94722	45812	54073	1.752	28267	3.351	14382	6.586	14382	6.586

Tabela 1.

Primer 2: Funkcija koja 4 puta rekurzivno poziva samu sebe:

```
(defun t4 (x)
    (if (= x 0)
        1
        (+
            (+ (t4 (- x 1))
                (t4 (- x 1)))
            (+ (t4 (- x 1))
                (t4 (- x 1))))))
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
5	1024	530	275	311	1.704	186	2.849	158	3.354	178	2.977
6	4096	2114	934	1203	1.757	657	3.218	382	5.534	382	5.534
7	16384	8450	3790	4772	1.771	2533	3.336	1.325	6.377	1325	6.377
8	65536	33797	15215	19045	1.775	10136	3.334	5.117	6.605	5117	6.605
9	262144	135189	60699	76138	1.776	40467	3.341	20.282	6.665	20282	6.665

Tabela 2.

Primer 3: Funkcija koja 5 puta rekurzivno poziva samu sebe:

```
(defun t5 ( x )
    (if (= x 0)
        1
        (+
            (t5 (- x 1))
            (+
                (+ (t5 (- x 1))
                    (t5 (- x 1)))
                (+ (t5 (- x 1))
                    (t5 (- x 1)))))))
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
4	625	308	110	207	1.488	204	1.51	347	0.888	477	0.646
5	3125	1533	659	811	1.89	574	2.671	740	2.072	1249	1.227
6	15625	7660	3406	3873	1.978	2141	3.578	1561	4.907	1464	5.232
7	78125	38294	16919	19160	1.999	9589	3.994	6772	5.655	6762	5.663
8	390625	191455	84759	95584	2.003	47090	4.066	32410	5.907	32290	5.929

Tabela 3.

Primer 4: Funkcija koja 6 puta rekurzivno poziva samu sebe:

```
(defun t6 ( x )
    (if (= x 0)
        1
        (+
            (+ (t6 (- x 1))
                (t6 (- x 1)))
        (+
            (+ (t6 (- x 1))
                (t6 (- x 1)))
```

```
        (+ (t6 (- x 1))
           (t6 (- x 1)))))))
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
4	1296	617	275	371	1.663	265	2.328	386	1.598	827	0.746
5	7776	3691	1593	2103	1.755	1164	3.171	819	4.507	1042	3.542
6	46656	22140	9668	12493	1.772	6685	3.312	3511	6.306	3511	6.306
7	279936	132837	58227	74837	1.775	39815	3.336	20076	6.617	20076	6.617

Tabela 4.

Primer 5: Rekurzivno nalazenje Fibonacijevih brojeva[2]

```
(defun fib ( x )
    (if (< x 2)
        x
        (+ (fib (- x 1)) (fib (- x 2)))))
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
10	55	65	54	49	1.327	36	1.806	26	2.5	22	2.955
15	610	710	385	502	1.414	337	2.107	201	3.532	138	5.145
16	987	1148	604	809	1.419	543	2.114	317	3.621	216	5.315
17	1597	1857	989	1306	1.422	874	2.125	506	3.67	343	5.414
18	2584	3004	1593	2111	1.423	1411	2.129	826	3.637	548	5.482
19	4181	4860	2637	3412	1.424	2279	2.132	1310	3.71	879	5.529
20	6765	7862	4230	5519	1.425	3684	2.134	2115	3.717	1416	5.552
21	10946	12721	6811	8927	1.425	5957	2.135	3418	3.722	2284	5.57
22	17711	20582	11041	14433	1.426	9476	2.172	5523	3.727	3688	5.58
23	28657	33301	17853	23353	1.426	15311	2.175	8932	3.728	5961	5.586
24	46368	53881	28839	37758	1.427	24761	2.176	14453	3.728	9639	5.59

Tabela 5.

[2] Ovaj algoritam je poznat kao najsporiji za nalaženje Fibonačijevih brojeva, i ne treba ga koristiti u praksi, osim u svrhe testiranja.

Primer 6: Neki problemi su takve prirode (mnogo ulazno-izlaznih operacija, veliki broj komunikacija, ili neko od ostalih tehničkih ograničenja transpjutera), da je njihovo izvršavanje neznatno brže, ili čak sporije na 17 transpjutera, u odnosu na izvršavanje na 1 transpjuteru. Primer je program koji formira "veliku" listu:

```
(define napravi (n)
( if (= n 0) (set lista (cons ´1 lista))
    (begin
     (napravi (- n 1))
     (napravi (- n 1))
     (napravi (- n 1))
     (napravi (- n 1))
     (napravi (- n 1)))))
(set lista ´())
```

X	Rez.	1 tr.	PC	3 tr.	ubrzanje	7 tr.	ubrzanje	15 tr.	ubrzanje	17 tr.	ubrzanje
2	-	53	25	63	0.841	63	0.841	63	0.841	63	0.841
3	-	293	137	344	0.852	344	0.852	344	0.852	344	0.852
4	-	1461	685	1721	0.849	1721	0.849	1721	0.849	1721	0.849
5	-	7338	3438	8633	0.85	8633	0.85	8633	0.85	8633	0.85

Tabela 6.

4.4. Ostale metode implementacije

Postoje različiti načini implementacije prevodilaca ili interpretera za funkcionalne programske jezike, na paralelnim računarskim sistemima. Mogu se podeliti u dve grupe ([21]):

(a) Paralelni jezici u kojima se paralelizam ostvaruje eksplicitno (explicitly parallel languages), preko konstrukcija jezika. Takvi programski jezici se lakše implementiraju ([21]), ali je programiranje na njima teže.

(b) Paralelni jezici u kojima se paralelizam ostvaruje implicitno (implicitly parallel languages). Prevodilac vodi sam računa o strategiji paralelnog izvršavanja, što olakšava programiranje na takvim programskim jezicima. Implementacija takvih programskih jezika je teža u odnosu na paralelne programske jezike u kojima se paralelizam ostvaruje eksplicitno (videti [21]).

U ovom radu je implementiran interpreter za programski jezik LISP, u kome se paralelizam ostvaruje implicitno.

Neki od načina za implementaciju funkcionalnih programskih jezika, su:

❶ Prevodjenje programa na medjujezik, koji je ASCII reprezentacija grafa objekata. Graf objekata opisuje složene izraze (iteracije, selekcije, ...), kao podgrafove datog grafa (opširnije videti u ([26] [28]).

❷ Podela problema rekurzivno na potprobleme (divide and conquer). U nekim implementacijama je takav pristup jedini moguć (videti [23]). Druge implementacije kombinuju takav pristup i neki drugi način ([22]). Sličan koncept je primenjen u ovom radu, ali se implementacija razlikuje.

❸ Prevodjenje programa u izvršni kod, koji ne zadovoljava zahteve brzine. Posle toga se, u procesu izvršavanja koda, automatski vrši analiza datog izvršnog koda, i poboljšavanje njegovih performansi. Prednost datog pristupa je u prilagodjavanju izvršnog koda, karakteristikama računara na kome se izvršava, bez promena programskog koda. Ovakav pristup ima više konkretnih načina implementacije ([24] [30]).

❹ Poboljšanje nekih implementacija uvodjenjem dodatnih optimizacija (videti [33]).

❺ Prevodjenje na medjujezik koji poseduje mali broj paralelnih konstrukcija (velike izražajne moći). Zatim se vrše analize datog koda i optimizuje komunikacija. Optimizacija obuhvata i nemogućnost formiranja velikih privremenih struktura podataka koje bi usporile izvršavanje. Dati pristup se može videti u [29].

❻ Nalaženje striktnih funkcija (kojoj nije definisana vrednost ukoliko je neki od argumenata nedefinisan), koje se mogu nezavisno (paralelno) izvršavati, u jezicima koje karakterišu tzv. "lenje" (lazy) gramatike. Jedna takva implementacija se može videti u [32].

❼ Markiranje nekih promenljivih, čija je vrednost potrebna datom procesoru, a izračunava je neki drugi procesor. Dati procesor nastavlja sa radom. Ukoliko je datom procesoru (ili nekom drugom) potrebna vrednost, a vrednost još nije izračunata, takav procesor se blokira. Procesor nastavlja sa radom, u trenutku kada je vrednost izračunata. Detaljnije o datoj implementaciji videti u [31].

Uporedna analiza nekih od načina implementacije, i poredjenje sa drugim metodima, može se videti u [27].

5. Zaključak

Čisti funkcionalni jezici imaju manje potrebe (i mogućnosti) za kontrolom računara na niskom nivou u odnosu na proceduralne programske jezike, pa su često bili zapostavljani. Međutim, masovnijom pojavom višeprocesorskih računara, javlja se potreba da se na nivou jezika rešavaju problemi komunikacije i sinhronizacije procesora. Pod ovim uslovima funkcionalni programski jezici se mnogo lakše implementiraju, nego proceduralni jezici iz sledećih razloga:

❶ Jezgro jezika (broj ključnih reči) je manje.

❷ Gramatika je konciznija, pa su konstrukcije uniformnije

❸ Raspored izvršavanja (izračunavanja) nije eksplicitan,

❹ Lakše se pišu rekurzivne funkcije,

❺ Ne postoje sporedni efekti.

Zbog svega ovoga funkcionalni programski jezici postaju sve popularniji.

Jedan od problema, koji nastaju sa masovnijom pojavom paralelnih računara, je nedostatak odgovarajućih programa. Ovaj problem se može rešavati na nekoliko načina:

(a) Nastaviti sa programiranjem na klasičnim sekvencijalnim računarima. U tom slučaju morali bismo se pomiriti sa slabijim performansama.

(b) Prilagoditi postojeće sekvencijalne programe izvršavanju u višeprocesorskim okolinama. Tada se, zbog same prirode rešenja, može dogoditi da program koji je davao odlične rezultate, u slučaju sekvencijalnog izvršavanja, posle prilagođavanja daje loše rezultate pri paralenom izvršavanju. Osim toga, prilagodjavanje je složen posao i često može biti teže od pisanja novog programa. Detaljnije o jednom takvom sistemu za prevodjenje sekvencijalnih programa u paralelne može se videti u [25].

(c) Napisati postojeće programe ponovo, tako da rade na višeprocesorskim računarima. Ovaj pristup, po pravilu, daje najefikasnije programe jer prevodilac ili interpreter ne može da reši probleme u paralelizaciji sekvencijalnog programa tako efikasno kao programer. Ozbiljan nedostatak ovog rešenja je što zahteva veoma mnogo programerskog rada.

(d) Podeliti ceo program na odredjene programske celine, zatim ih rasporediti kao procese na datim procesorima. Pri tome se javlja problem konkretnog rasporedjivanja procesa na višeprocesorskom sistemu. Nalaženje takvog optimalnog rasporeda je, na žalost, NP problem (opširnije o NP problemima videti u [5] [20]), a nalaženje neoptimalnog rasporeda (heuristički), može dati loš rezultat.

(e) Konstruisati interpreter (kao u ovom radu), ili prevodilac, koji implicitno rešava probleme komunikacije i sinhronizacije. Program koji se izvršava je isti kao i za sekvencijalni računar, a izvršavanje na paralelnom računaru je transparentno za korisnika. Sve probleme oko komunikacije i sinhronizacije rešava sam prevodilac ili interpreter. Ovakvo rešenje nije efikasno kao rešenje navedeno pod c), ali oslobađa programera velikog posla.

(f) Primeniti neku od preostalih ideja, ili kombinovati postojeće varijante.

U ovom radu je primenjena varijanta (e), za koju se čini da je najopštiji, mada ne i najefikasniji način rešavanja navedenog problema. Pri ovakvom pristupu korisnički program je isti kao i za sekvencijalne računare. To znači da se svi postojeći programi pisani na odgovarajućem programskom jeziku mogu direktno preneti na paralelni računar. Isto tako programer i ubuduće može za razvoj i testiranje programa da koristi jednoprocesorski računar i pripadajuće programske alate, a da finalnu verziju izvršava na paralelnom računaru.

U realizaciji interpretera je primenjena hijerarhijska arhitektura (binarno drvo). Zato se može desiti da neki procesor radi dok njegov naslednik čeka, jer ne može dobiti podatke za izračunavanje. Čekanje bi se moglo smanjiti ako bi se primenila neka složenija arhitektura veza, ali bi se problem sinhronizacije procesora veoma iskomplikovao, što bi otežalo realizaciju. Osim toga, više vremena bi se trošilo na međuprocesorsku komunikaciju, pa bi se moglo desiti da i pored veće iskorišćenosti procesora, rezultati ne budu znatno bolji, ili da čak budu gori.

Dalje poboljšanje ove implementacije je moguće u dva pravca.

❶ Ugradjivanje novih funkcija do Common LISP standarda. Problem je tu više praktične prirode zbog brojnosti tih funkcija (preko hiljadu), nego zbog teškoća u dogradjivanju.

❷ Realizacija intepretera sa složenijom arhitekturom veza (u odnosu na binarno drvo). Takav pristup bi, sa novom generacijom transpjutera T9000™, mogao dati još bolje rezultate u odnosu na T800™. Jer sa 16 ulaznih i 16 izlaznih kanala može se formirati mreža transpjutera koja bi bila bitno složenija od binarnog drveta (naročito za veći broj transpjutera), pa bi vremenski dobici mogli biti značajniji.

Implementacijom na višeprocesorskom računaru sa 17 transpjutera, postiže se (u problemima sa dosta računanja) značajno ubrzanje (do 7 puta) u odnosu na izvršavanje na jednom transpjuteru. Zbog prirode problema koji se rešava, kao i zbog tehničke organizacije transpjuterskog sistema, praktično se ne može postići da ubrzanje bude srazmerno broju upotrebljenih transpjutera. Međutim, i pored relativne neefikasnosti, čini se da je ovaj pravac razvoja računarske tehnologije najperspektivniji u smislu poboljšavanja performansi.

Već sada trend paralelizacije daje rezultate vredne pažnje, a iz razloga iznetih u uvodnom poglavlju, može se očekivati da će se u vrlo skoroj budućnosti ovaj trend izdvojiti od ostalih po svojim rezultatima.

U prilog ovom mišljenju konstatujmo da cene pojedinačnih procesora padaju, pa će konfiguracije sa veoma velikim brojem procesora biti znatno pristupačnije.

Dodatak A Transpjuteri kao višeprocesorski računari

A.1. Tehničke karakteristike

Traspjuteri pripadaju klasi MIMD MP paralelnih računarskih sistema. Teorijski se može izabrati proizvoljna arhitektura veza, medjutim postoje neka vrlo stroga tehnička ograničenja.

Svaki transpjuter T800™ radi na 20Mhz, i poseduje po 1MB linearne memorije, celobrojnu i racionalnu aritmetiku.

Svaki procesor poseduje 4 ulazna i 4 izlazna kanala za komunikaciju sa ostalim transpjuterima. Svaki transpjuter može resetovati najviše 2 druga transpjutera (opširnije videti [12]). Postoji i ograničenje, da ukoliko prvi transpjuter komunicira sa drugim transpjuterom kanalom broj i ($0 \leq i \leq 3$), tada i drugi takodje mora komunicirati sa prvim transpjuterom pomoću svog i-tog kanala. Ovo poslednje rezultuje nemogućnošću nekih arhitektura (hiperkocka sa 16 čvorova). Arhitektura veza se zadaje u datoteci sa ekstenzijom .NIF.

Vreme potrebno za prenos jednog (celobrojnog) podatka je oko 4 puta duže, od vremena za aritmetičku operaciju (sabiranje ili množenje nad tim podatkom). To se može videti iz tabele 7:

Broj podataka	Prenos podataka	Sabiranje (celobrojno)	Množenje (celobrojno)
10 000	208 ms	54 ms	56 ms
100 000	2202 ms	544 ms	565 ms
200 000	4488 ms	1089 ms	1134 ms
1 000 000	22654 ms	5478 ms	5685 ms

Tabela 7.

Napomena: Testirana je brzina prenosa podataka izmedju dva susedna transpjutera (slanje sa 2. transpjutera na 1.). Pri prevodjenju i izvršavanju su korišćene podrazumevane (default) vrednosti parametara za komunikaciju izmedju transpjutera (videti [13]). Iste (podrazumevane) vrednosti su korišćene i pri testiranju brzine interpretera. Odstupanje od izmerenih vrednosti vremena je najviše±5 ms

Rezultati u tabeli 7. (za 1 000 000 podataka) dobijeni su pomoću programa na slici 2.

Deo koji se izvršava na prvom transpjuteru:

```
#include "stdio.h"
#include "conc.h"
#define N 200000

int i,j,k,vr1,vr2,vr,s;
int a[N];

main()
{
    ProcToHigh();
    vr1 = Time();
    for(k=0; k<5; k++)
     for(i=0; i<N; i++)
       {
         j = ProcAlt(LINK1IN,0);
         a[i] = ChanInInt(LINK1IN);
       }
    vr2 = Time();
    vr = vr2 - vr1;
    printf("Prenos trajao %d milionitih",vr);
    s = 0;
    vr1 = Time();
    for (k=0;k<5; k++)
    for (i=0; i<N; i++) s = s+a[i];
    vr2 = Time();
    vr = vr2 - vr1;
    printf("Sabiranje trajalo %d milionitih",vr);
    s = 1;
    vr1 = Time();
    for(k=0; k<5; k++)
     for (i=0; i<N; i++) s = s*a[i];
    vr2 = Time();
    vr = vr2 - vr1;
    printf("Mnozenje trajalo %d milionitih",vr);
}
```

Deo koji se izvršava na ostalim transpjuterima:

```
#include <conc.h>
#define N 200000

int i,k;
int a[N];
main()
{
for(i=0; i<N; i++) a[i]= i;
for(k=0; k<5; k++)
for(i=0; i<N; i++) ChanOutInt(LINK1OUT,a[i]);
}
```

Slika 2.

A.2. Programski jezici za transpjutere

Za transpjutere postoje 2 vrste viših programskih jezika:

❶ Jezici specijalno konstruisani za paralelne računare (OCCAM)

❷ Klasični programski jezici, sa dodatkom procedura za rad sa procesima i komunikaciju izmedju procesora (Paralelni C, Paralelni Pascal, Paralelni Fortran, itd).

Implementacija LISP-a na transpjuterima u ovom radu je izvršena u paralelnom C-u.

A.3. Paralelni C

Paralelni C je programski paket koji se sastoji od C preprocesora, prevodioca za C, assemblera, linkera, loadera, i biblioteka. Izvršava se pod DOS operativnim sistemom (postoje verzije i za ostale operativne sisteme, kao na primer, za UNIX).

PP C je standardni preprocesor, koji vrši obradu # direktiva. Ulaz mu je datoteka sa nastavkom .C, a izlaz sa nastavkom .PP. O sintaksi opširnije pogledati [7] .

TCX cross compiler je prevodilac za paralelni C. Zahteva 512KB memorije, i ne koristi nikakve privremene datoteke, već prevodjenje vrši u memoriji. Skoro u potunosti odgovara ANSI C standardu, osim što ne podržava bit-polja, i long-double tip podataka. Bitna je opcija **-p8,** koja se koristi za generisanje T800™ koda (po defaultu je generisanje koda za T414). Rezultujući kod je asemblerski kod (.TAL) koji odgovara datom izvornom kodu. O ostalim opcijama detaljnije pogledati u [8].

Transputer ASseMbler (TASM) prevodi dati asemblerski kod u mašinski kod (.TRL). Zahteva najmanje 256KB memorije. Detaljnije o asembleru se može videti iz [9].

TLNK linker vrši povezivanje mašinskog koda sa spoljnim bibliotekama. Ograničenje je maksimalan dozvoljen broj od 2048 spoljnih simbola i ukupan broj simbola od najviše 5450. Informacije je moguće uneti sa tastature, ili iz datoteke sa nastavkom (.LNK). Detaljnije o opcijama linkera videti u [10] .

TLIB koristi se za za manipulaciju spoljnim izvršnim bibliotekama. Date su verzije biblioteka za T800™ gde su **double** brojevi sa 64 bita (t8lib.tll), odnosno gde su 32 bita (t832lib.tll). Moguće je pravljenje i manipulacija sa korisničkim bibliotekama (videti [13]).

Za učitavanje programa na transpjutere i njihovo izvršavanje služe LD-ONE i LD-NET. LD-ONE vrši učitavanje programa na prvi transpjuter i njegovo izvršavanje na njemu (detaljnije u [11]).

LD-NET učitava Network Information File (.NIF), resetuje osnovni čvor (1. transppjuter), učitava i izvršava primarni i sekundarni bootstrap na 1. transpjuteru.

Zatim 1. transpjuter vrši reset transpjutera na sistemskom (system) i podsistemskom (subsystem) izlazu. Posle toga svaki od transpjutera vrši prethodne operacije na svojim potomcima (primarni i sekundarni bootstrap, i reset na sistemskom i podsistemskom kanalu).

Dalje se šalju svi programi svakom od transpjutera. Svaki transpjuter dobija sve programe, iako se na njemu izvršava samo jedan, ali se u ovom slučaju ne javlja problem oko toga kom transpjuteru treba poslati koji program, već se šalju svi. Svaki transpjuter šalje programe po svojim sistemskim i podsistemskim vezama.

Kada su svi programi učitani, računar domaćin (PC koji se koristi za vezu sa transpjuterima) šalje komandu EXECUTE, svim čvorovima, koja startuje izvršavanje na svakom od čvorova. Računar domaćin tada preko specijalnog I/O drajvera (CIO ili TIO) izvršava ulazno-izlazne operacije sa 1. transpjutera. Ostalim transpjuterima nije dozvoljen ni ulaz ni izlaz. Opširnije o ovome videti u [12] .

Biblioteci standardnih C procedura koje su predvidjene ANSI standardom (osim već pomenutih) pridodate su procedure za rad sa procesima po Jeffrey Mock metodu, Fork/Join metodu (kao u UNIX-u), i procedure za komunikaciju izmedju transpjutera (videti detaljnije u [6] [14]).

A.4. Procedure za komunikaciju po Jeffrey Mock metodu

GetHiPriQ() - Očitava pokazivače na red čekanja višeg prioriteta.

GetLoPriQ() - Očitava pokazivače na red čekanja nižeg prioriteta.

HSemP() - Procedura za rad sa semaforima. Zaključava tekuću operaciju za korišćenje samo na tekućem procesu, dok ostali procesi koji eventualno žele da izvrše datu operaciju čekaju. Radi i za procese sa različitim prioritetima.

HSemV() - Procedura za rad sa semaforima. Otključava tekuću operaciju i dozvoljava pristup i ostalim procesima. Radi i za procese sa različitim prioritetima.

ProcAfter() - Nastavlja izvršavanje tekućeg procesa posle zadatog vremena.

ProcAlloc() - Alocira proces, pri tome koristi proceduru malloc(). Svaki proces mora se alocirati da bi se mogao koristiti.

ProcCall() - Ako je neka procedura često korišćena, i manja je od 512B, može se smestiti u internu memoriju transpjutera. Tada se izvršava višestruko brže, što ubrzava ceo program.

ProcFree() - Oslobadja memoriju koju je zauzeo proces.

ProcGetPriority() - Vraća prioritet procesa. Ako je proces bio višeg prioriteta vraćena vrednost je 0, a za proces nižeg prioriteta 1.

ProcInit() - Inicira dati proces.

ProcPar() - Startuje grupu procesa. Parametri su procesi, a lista parametara završava se nulom. Kontrola se vraća tekućem procesu kada se svi novi procesi završe.

ProcParam() - Menja parametre procesa. Proces mora prethodno biti inicijalizovan pomoću ProcAlloc() ili ProcInit(). Memorijski prostor ne sme biti u tom trenutku u upotrebi od datog procesa, jer su tada rezultati nepredvidljivi.

ProcParList() - Isto kao i ProcPar(), samo se zadaje pokazivač na listu procesa.

ProcPriPar() - Isto kao ProcPar(), ali startuje izvršavanje dva procesa prvog na višem, a drugog na nižem prioritetu.

ProcReschedule() - Blokira tekući proces i postavlja ga na kraj reda čekanja za tekući prioritet. To je neophodno ako proces čeka neki resurs.

ProcRun() - Izvršava dati proces. Proces "roditelj" gubi kontrolu nad kreiranim procesom (osim pomoću eksplicitne komunikacije izmedju procesa). Oba procesa nastavljaju da se izvršavaju. Prioritet kreiranog procesa je isti kao i tekućeg procesa.

ProcRunHigh() - Isto kao i ProcRun(), ali se kreirani proces izvršava na višem prioritetu bez obzira na kom prioritetu je bio "roditeljski" proces.

ProcRunLow() - Isto kao i ProcRun(), ali se kreirani proces izvršava na nižem prioritetu bez obzira na kom prioritetu je bio "roditeljski" proces.

ProcStop() - Proces izbacuje sa reda čekanja (ukida ga). Više ga nije moguće vratiti. Ekvivalentna je PStop() funkciji kod Fork/Join modela (videti poglavlje o Foork/Join modelu).

ProcToHigh() - Postavlja prioritet procesa na viši nivo.

ProcToLow() - Postavlja prioritet procesa na niži nivo.

ProcWait() - Čeka odredjeni interval vremena. Jedinični interval čekanja zavisi od tekućeg prioriteta pod kojim se proces izvršava.

SemP() - Zaključava semafor. Radi isto što i HSemP(), ali radi samo sa procesima istog prioriteta.

SemV() - Otključava semafor. Ista je kao HSemV(), ali radi samo sa procesima istog prioriteta.

SetHiPriQ() - Postavlja pokazivače na red čekanja višeg prioriteta.

SetLoPriQ() - Postavlja pokazivače na red čekanja nižeg prioriteta.

SetTime() - Postavlja vreme. Osnovna jedinica vremena zavisi od toga na kom prioritetu se izvršava dati proces.

Time() - Očitava trenutno vreme. Vraćena vrednost zavisi od toga na kom prioritetu se izvršava dati proces.

SemAlloc() - Alocira semafor. Svaki semafor mora biti alociran eksplicitno ovom procedurom ili pri deklarisanju.

SemFree() - Oslobadja memoriju koju je zauzeo semafor.

Konstanta **_node_number** označava redni broj transpjutera

A.5. Fork/Join procedure za rad sa procesima

Ravnopravno Jeffrey Mock metodu za rad sa procesima koriste se procedure za rad sa procesima po Fork/Join metodu. Programski kod na UNIX C-u za rad sa procesima je potpuno kompatibilan. Kod je kompatibilan i prenosiv za procese na istom transpjuteru, dok je komunikacija izmedju transpjutera realizovana preko kanala.

PFork() - Startuje proces. Parametri su ForkBlk struktura i proces deskriptor. Kreirani proces radi na istom prioritetu kao "roditeljski" proces.

PForkHigh() - Isto kao PFork(), ali kreirani proces se izvršava na višem prioritetu.

PForkInit() - Inicijalizuje ForkBlk sturkturu. Parametri su data ForkBlk struktura i broj procesa koji se startuju.

PForkLow() - Isto kao PFork(), ali kreirani proces se izvršava na nižem prioritetu.

PHalt() - Ukida tekući proces, ali čuva strukturu procesa (PDes), pa on može biti kasnije ponovo startovan sa PRun().

PJoin() - Blokira sve dok svi procesi koji odgovaraju datoj ForkBlk strukturi ne završe sa radom.

PRun() - Startuje proces pridružen datoj PDes strukturi. Struktura prethodno mora biti inicijalizovana pomoću PSetup().

PSetup() - Inicijalizuje memorijski prostor za izvršavanje procesa i proces deskriptor.

PStop() - Ukida tekući proces. Ako nije snimljen proces deskriptor nemoguće je ponovo startovati dati proces.

A.6. Procedure za rad sa kanalima pri komunikaciji izmedju transpjutera

Date su i procedure za rad sa kanalima za komunikaciju ([6] [14]). Kanal može biti hardverski, za komunikaciju izmedju različitih procesora, ili softverski za komunikaciju izmedju procesa na istom transpjuteru:

ChanAlloc() - Alocira softverski kanal za komunikaciju izmedju procesa na istom transpjuteru. Kanali za komunikaciju izmedju transpjutera su hardverski i nije im potrebno alociranje.

ChanFree() - Oslobadja memoriju koju je zauzeo dati softverski kanal.

ChanIn() - Prima blok bajtova sa ulaznog kanala. Argumenti su pokazivač na ulazni kanal, pokazivač na blok bajtova, i ceo broj koji označava veličinu bloka. Ne vraća nikakvu vrednost.

ChanInChanFail() - Koristi se za ispitivanje greški na kanalu, i ne treba je koristiti za prijem podataka već ChanIn(), ChanInChar() ili ChanInInt().

ChanInChar() - Prima znak sa ulaznog kanala. Argument je pokazivač na kanal, a vraća kao rezultat dati znak.

ChanInInt() - Prima ceo broj sa ulaznog kanala. Argument je pokazivač na ulazni kanal, a vraća kao rezultat dati ceo broj.

ChanInTimeFail() - Kao i ChanInChanFail() koristi se za za ispitivanje greški na ulaznom kanalu.

ChanOut() - Prosledjuje blok bajtova na izlazni kanal. Argumenti su pokazivač na izlazni kanal, pokazivač na blok bajtova, i ceo broj koji označava veličinu bloka. Ne vraća nikakvu vrednost.

ChanOutChanFail() - Kao i ChanInChanFail(), ali za izlazni kanal.

ChanOutChar() - Prosledjuje znak na dati izlazni kanal. Argumenti su pokazivač na kanal i znak, a ne vraća nikakvu vrednost.

ChanOutInt() - Prosledjuje ceo broj na dati izlazni kanal. Argumenti su pokazivač na kanal i znak, a ne vraća nikakvu vrednost.

ChanOutTimeFail() - Kao i ChanInTimeFail(), ali za izlazni kanal.

ChanReset() - Ukoliko dodje do problema na kanalu, prazni dati kanal.

ProcAlt() - Argumenti su ulazni kanali (lista kanala je terminirana nulom). Čeka dok neki od njih ne bude spreman za ulaz, i vraća njegov indeka u listi argumenata. Tek tada nastavlja dalje sa radom.

ProcAltList() - Isto kao i ProcAlt(), ali argument nije cela lista ulaznih kanala već pokazivač na nju.

ProcSkipAlt() - Isto kao i ProcAlt(), ali ne čeka ukoliko nijedan kanal u listi nije spreman za ulaz, već nastavlja sa izvršavanjem programa.

ProcSkipAltList() - Isto kao i ProcSkipAlt(), ali argument nije cela lista ulaznih kanala već pokazivač na nju.

ProcTimerAlt() - Isto kao ProcAlt(), ali čeka samo do datog vremena, a zatim ukoliko nijedan od kanala nije spreman za ulaz nastavlja sa izvršavanjem programa.

ProcTimerAltList() - Isto kao ProcAltList(), ali čeka samo do datog vremena, a zatim ukoliko nijedan od kanala nije spreman za ulaz nastavlja sa izvršavanjem programa.

Konstante:

_boot_chan_out - pokazivač na izlazni kanal prema transpjuteru koji vrši resetovanje tekućeg transpjutera.

_boot_chan_in - pokazivač na ulazni kanal od transpjutera koji vrši resetovanje tekućeg transpjutera.

LINK0OUT

LINK1OUT

LINK2OUT

LINK3OUT

LINK0IN

LINK1IN

LINK2IN

LINK3IN

predstavljaju adrese 4 izlazna i 4 ulazna hardverska kanala na svakom od transpjutera. Služe za komunikaciju izmedju transpjutera.

Dodatak B Detaljan opis implementacije

U ovom dodatku je dat detaljniji opis implementacije u odnosu na poglavlja 3. (Implementacija jezgra LISP-a na jednoprocesorskim računarima) i 4. (Implementacija LISP-a na transpjuterima).

B.1. Opis implementacije na jednoprocesorskim računarima

B.1.1 Definicije

Ova celina sadrži deklaracije svih konstanti i tipova u programu.

Konstante:

duzimena - Maksimalna dužina svakog imena.

maximena - Maksimalan broj imena.

maxulaz - Maksimalna dužina ulaza.

prompt - Osnovni odzivni znak (prompt).

prompt2 - Pojavljuje se ukoliko tekst prelazi okvire jednog reda, pa se potpuno slobodno može nastaviti u sledećem redu.

komentar - Oznaka za komentar posle koga se svi znakovi u tekućem redu ignorišu (tretiraju kao komentar), i ukoliko tekst nije završen, prelazi na obradu sledećeg reda. U suprotnom se završava ulazna operacija.

tabkod - Ascii kod za tab.

Definicije tipova (struktura):

Struktura **izrazstr** predstavlja izraz (realizovan kao unija) i može biti S-izraz, promenljiva, funkcija, ili lista izraza što zavisi od polja **tipizr** u kome se beleži tip izraza.

Struktura **sizrstr** predstavlja S-izraz (takodje realizovan kao unija) i može biti prazan, brojni, simbolički S-izraz, ili lista S-izraza. To zavisi od polja **tipsizr** u kome se memoriše tip S-izraza.

Definicije svih funkcija sadrži struktura **deffunstr**. Polja predstavljaju ime funkcije (odnosno redni broj u nizu imena), pokazivač na formalne argumente, telo funkcije (takodje pokazivač) i pokazivač na sledeću definiciju funkcije.

Listu imena (rednih brojeva u nizu imena) funkcija, odnosno promenljivih sadrži **listaimstr**. Polja su ime i pokazivač na listu preostalih imena.

Tekuće vrednosti odgovarajućih promenljivih sadrži **stanjestr**. Polja su lista imena promenljivih i lista tekućih vrednosti.

Struktura **listaizrstr** je lista izraza. Prvi element je izraz, a drugi pokazivač na listu preostalih izraza.

Listu vrednosti memoriše **listavrstr**. Polja su pokazivač na S-izraz, i pokazivač na listu preostalih vrednosti.

Odgovarajući pokazivači na njih su: **izraz, listaizr, deffun, stanje, sizr, listaim** i **listavr.**

B.1.2 Deklaracije promenljivih

svedeffun - Sve korisničke funkcije (definicije).

ukupnostanje - Trenutne vrednosti svih promenljivih.

tekizr - Predstavlja tekući izraz.

S-izrazi **nula_vred** i **tacna_vred** predstavljaju prazan S-izraz (koji predstavlja i logičko $\perp$) odnosno tačnu vrednost (logičko **T**).

ulazpod - String u kome memorišemo ulazne podatke pre njihove obrade.

duzulaza - Označava koliko znakova ima string **ulazpod.** Maksimalna vrednost je predstavljena konstantom **maxulaz**.

poz - Tekući znak do koga smo stigli u obradi ulaznih podataka. Kada mu vrednost bude jednaka **duzulaza**, završili smo obradu tekućeg ulaza (**ulazpod**) i prelazimo na postavljanje nove funkcije ili promenljive (ako je u pitanju definicija funkcije ili promenljive), odnosno na izračunavanje vrednosti (poziv funkcije).

nizimena - Tekući niz imena svih ugradjenih i korisničkih funkcija i promenljivih. Dužina svakog imena je data konstantom **duzimena**, a maksimalan broj imena konstantom **maximena.**

brimena - Tekući broj imena niza **nizimena**.

brugradjenih - Predstavlja broj ugradjenih funkcija. Imena ugradjenih funkcija su na pocetku niza **nizimena** a zatim slede imena promenljivih i korisničkih funkcija.

krajrada - Indikator kada treba završiti izvršavanje programa. Postavlja se na **1** ako se na ulazu pojavi string **"quit"**.

argstek - Stek za smeštanje argumenata pri izračunavanju vrednosti funkcije.

vrhargstek - Vrh steka za smeštanje argumenata.

slobsizr - Pokazivač na listu slobodnih S-izraza.

B.1.3. Upravljanje memorijom

initmem() - Alocira memoriju za sve S-izraze. Iz date liste slobodnih S-izraza, se dodeljuju S-izrazi. Iskorišćeni S-izrazi vraćaju se u listu slobodnih S-izraza.

alocsizr() - Dodeljuje S-izraz iz liste slobodnih S-izraza.

vratisizr() - Iskorišćeni S-izraz vraća u listu slobodnih S-izraza.

povrefbr() - Povećava refbr polje datog S-izraza, odnosno povećava broj referenci na njega. To se dešava u slučaju dodatnog korišćenja datog S-izraza.

smarefbr() - Smanjuje refbr polje datog S-izraza, odnosno smanjuje broj referenci na njega,

oslrefbr() - Smanjuje refbr u svim S-izrazima osim datog,

iniargstek() - Postavlja stek za argumente,

staviarg() - Stavlja jedan S-izraz na argument stek,

uzmivrharg() - Vraća vrh argument steka,

uzmidovrharg() - Vraća prvi S-izraz ispod vrha.

i**zbaciarg()** - Izbacuje dati broj argumenata sa argument steka, počev od vrha.

B.1.4. Podrška osnovnim tipovima podataka

Ova celina sadrži procedure za podršku osnovnim tipovima podataka:

napvrizr() - Rezerviše memoriju za vrednosni izraz.

napprizr() - Alocira memoriju za izraz tipa promenljive.

napfunizr() - Služi za rezervisanje memorije za izraz tipa funkcije.

naplistizr() - Koristi se za alociranje memorijskog prostora za izraz koji predstavlja listu izraza. Vraća pokazivač na datu listu izraza.

naplistaizr() - Služi za alociranje memorije koju koristi lista izraza.

naplistaim() - Alocira memoriju za listu imena (listu rednih brojeva u **nizimena**).

naplistavr() - Rezerviše memoriju za listu vrednosti.

napstanje() - Služi za alokaciju memorije koju koristi **stanje**.

duzlistaim() - Vraća dužinu liste imena.

duzlistavr() - Vraća dužinu liste vrednosti.

B.1.5. Operacije nad imenima funkcija

U ovoj celini su date procedure za operacije nad imenima funkcija (nalaženje definicije, dodavanje nove definicije, postavljanje imena funkcija, dodavanje novog imena funkcije, štampanje imena funkcije i nalaženje rednog broja operacije). Procedure su:

uzmideffun() - Nalazi definiciju date funkcije **fime** u strukturi **svedeffun**.

novadeffun() - Dopunjava strukturu **svedeffun** definicijom nove funkcije. Argumenti su ime funkcije **fime**, formalni argumenti **nl** (tipa **listaim**), i telo funkcije **e**.

postaviim() - Inicijalizuje globalne promenljive:

❶ ***svedeffun*** na NULL.

❷ ***nizimena*** imenaima ugradjenih funkcija

❸ ***brimena*** i ***brugradjenih*** na broj ugradjenih funkcija.

novoime() - Dodaje novo ime funkcije u **nizimena**.

stamime() - Štampa ime **im**. To može biti ime funkcije ili ime promenljive.

nalaziop() - Na osnovu rednog broja imena **oper** nalazi vrstu operacije.

B.1.6. Ulaz podataka i definicija korisničkih funkcija

Sledeće procedure obradjuju učitavaju ulazne podatke, i smeštaju ih u niz **ulazpod. O**bradom datog niza, u slučaju definicije funkcije, izdvajaju odgovarajuće strukture podataka. Procedure su:

izdvsizr() - Izdvaja S-izraz iz niza **ulazpod**.

izdvlistsizr() - Iz niza **ulazpod** izdvaja listu S-izraza.

izdvizr() - Iz niza **ulazpod** izdvaja izraz.

izdvlistizr() - Koristi se za izdvajanje liste izraza.

izdvceob() - Izdvaja ceo broj iz niza **ulazpod**.

izdvsimb() - Iz niza **ulazpod** izdvaja simbol.

izdvime() - Služi za izdvajanje imena (redni broj u **nizimena**).

izdvlistim() - Koristi se za izdvajanje liste imena.

izdvfunkc() - Služi za izdvajanje definicije funkcije (ime, listu formalnih parametara i telo funkcije) iz niza **ulazpod**.

izdvvr() - Služi da izdvoji vrednost iz niza **ulazpod**.

Sledeće procedure učitavaju ulazne podatke i smeštaju ih u niz **ulazpod**:

sledkar() - učitava po jedan znak i smešta ga u niz **ulazpod**.

ucitblok() - učitava blok podataka zaključno sa znakom "**)**".

ucitulaz() - učitava deo ulaznog niza **ulazpod**.

ucitaj() - učitava ceo ulazni niz u datom trenutku.

Ostale procedure u ovoj celini su:

izbpraz() - Ignoriše prazne znakove (blanko) u nizu **ulazpod** i prelazi na prvi sledeći neprazan (nonblank) znak.

ististr() - Ispituje da li je string **nm** jednak nizu znakova **ulazpod[s]** do **ulazpod[s+duzina]**

dalije_cifra() - Ispituje da li je odgovarajući znak u **ulazpod** cifra?

dalije_broj() - Ispituje da li niz znakova počev od datog predstavlja dekadni broj?

dalije_delim() - Ispituje da li je dati znak delimiter?

ofsetprom() - Nalazi datu promenljivu (ime) iz liste imena.

prostlizrprom() - Primenjuje **postizrprom()** na svaki izraz u listi izraza.

postizrprom() - Postavlja ofset u listi imena.

B.1.7. Postavljanje parametara

Procedure u ovoj celini postavljaju ili ispituju parametre funkcija ili promenljivih. To su:

prazst() - vraća pokazivač na prazno stanje. Koristi se pri inicijalizaciji stanja.

postprom() - postavlja vrednost promenljive **im** na vrednost **n** u stanju **rho**. **rho** proširujemo novom promenljivom i njenom vrednošću.

nadjiprom() - nalazi promenljivu **im** u stanju **rho**. Ako data promenljiva ne postoji vraća NULL.

dodprom() - dodeljuje (postojećoj) promenljivoj **im** novu vrednost **n** u stanju **rho.**

vrprom() - vraća trenutnu vrednost promenljive **im** u stanju **rho**.

dalije_prom() - da li je dato ime promenljiva ili ne?

B.1.8. Manipulacija S-izrazima

Ova celina sadrži procedure za manipulaciju S-izrazima i nalaženje rezultata nekih operacija. Date procedure su:

stamsizr() - Štampa dati S-izraz.

dalije_tacan() - Da li je dati S-izraz tačan (različit od praznog S-izraza)?

primoper() - Primenjuje ugradjenu binarnu operaciju **op** (+,-,*,/) na argumente **n1** i **n2**. Rezultat prosledjuje (po adresi) argumentom **rezultat.**

primrel() - Primenjuje ugradjenu binarnu relaciju **op** (<,>) na argumente **n1** i **n2**. Istinitosnu vrednost prosledjuje argumentom rezultat.

primvrop() - Ispituje da li je **op** +,-,*,/,<, >. Ako jeste poziva odgovarajuću funciju (**primoper()** ili **primrel()**). Inače primenjuje datu operaciju (**cons, car, cadr,** =), ili štampa dati S-izraz.

B.1.9. Izračunavanje vrednosti izraza

Data celina je najbitnija, i procedure za izračunavanje vrednosti izraza, ili liste izraza:

izracunaj() - Za dati izraz **e** i stanje **rho**, nalazi vrednost datog izraza. Vraća vrednost kao S-izraz.

izraclistu() - Za datu listu izraza **el** i stanje **rho** izračunava vrednost. Rezultat vraća kao listu vrednosti.

primkonop() - Primenjuje kontrolnu operaciju (**if, cond, while, setq, begin**) na argumente **arg** tipa **listaizr** (lista izraza) pri stanju **rho**.

primkorop() - Primenjuje korisničku funkciju **im** na **stvparam** (stvarna vrednost parametara koji zamenjuju fiktivne parametre) koji je tipa **listavr** (lista vrednosti).

B.1.10. Glavni program

Ova celina sadrži samo dve procedure:

uradi()

❶ Poziva **ucitaj()** za učitavanje ulaznih podataka,

❷ Ispituje da li je nastupio kraj (**"quit"** kao ulazni podatak),

❸ Da li je u pitanju definicija korisničke funkcije (**"defun"**)? Tada izdvaja definiciju date funkcije, smešta je u **svedeffun**, a ime smešta u listu imena.

❸ Ili treba izračunati vrednost neke funkcije odnosno promenljive? Tada joj izračunava vrednost.

❹ Vrati se na korak ❶.

main() - postavlja početne vrednosti pa zatim poziva funkciju **uradi()**.

B.2. Opis implementacije na višeprocesorskim računarima

Kao osnova za implementaciju na transpjuterima je iskorišćena implementacija na jednoprocesorskim (PC) računarima. Uz sitne promene u prethodnim celinama, dodate su dve nove celine:

❶ Prenos argumenata i

❷ Kontrolni deo.

Postoje dva dela programa:

① Prvi deo se izvršava na prvom transpjuteru i sadrži sve što sadrži i verzija za PC računare uz odgovarajuće dodatne celine (❶ i ❷).

② Drugi deo se izvršava na ostalim transpjuterima (osim prvog), i osiromašena je za one procedure koje se ne mogu izvršiti na njima (ulazno-izlazne operacije i ostale procedure sličnog tipa).

B.2.1. Deo koji se izvršava na prvom transpjuteru

Promene u ovom delu, u odnosu na verziju na jednokorisničkim računarima, su samo u celinama **Definicije**, **Deklaracije promenljivih**, i Izračunavanje vrednosti izraza. Dodate su neke definicije i deklaracije dodatnih promenljivih. Nove celine koje su specifične za implementaciju na transpjuterima su:

❶ Prenos argumenata

❷ Kontrolni deo

U celini Izračunavanje vrednosti izraza nedostaje procedura **izraclistu(),** koja se modifikovana nalazi u Kontrolnom delu, a Glavni program kao celina ne postoji, već se sadrži u celini Kontrolni deo.

Ostale celine su nepromenjene, u odnosu na PC verziju.

B.2.1.1. Izmene u prethodnim celinama

Konstante:

BROJ_NASL - Broj naslednika svakog procesora (u ovom slučaju je to kod binarnog drveta jednako 2).

BROJ_TRANSP - Ukupan broj transpjutera (u ovom slučaju 17)

IMA_NASL - Koliko prvih transpjutera ima naslednike (u tekućem slučaju 8)

MAXST - Maksimalna vrednost steka za prosledjivanje vrednosti funkcije "roditeljskim" procesorima.

Tipovi podataka:

akcija - nabrojivi tip čiji su elementi:

1 ***RACUNAJ*** - Ovu poruku šalje "roditeljski" procesor "nasledniku", ukoliko mu šalje funkciju na izračunavanje. Naravno pre toga "naslednik" mora biti slobodan (niz **zauzet** je nula). Posle ove poruke "roditeljski" procesor šalje argumente tipa **izraz** i **stanje** pa zatim definiciju funkcije tipa **deffun**. Procesor "naslednik" za to vreme prima argumente, a zatim vrši izračunavanje vrednosti date funkcije. Izračunatu vrednost smešta na svoj stek za komunikaciju, a zatim šalje poruku "roditeljskom" transpjuteru da je postao slobodan.

2 ***ZAHTEV*** - Kada je "roditeljskom" procesoru potrebna vrednost funkcije koju je slao na izračunavanje jednom od "naslednika", on šalje ovu poruku datom "nasledniku".

3 ***VREDNOST*** - Kada "naslednik" dobije prethodnu poruku on šalje "roditeljskom" procesoru ovu poruku. Posle toga, sa svog steka za komunikaciju uzima prethodno izračunatu vrednost funkcije, koja mu je prosledjena na računanje od "roditeljskog" transpjutera. Zatim tu vrednost šalje "roditeljskom" procesoru

4 ***SLOBODNO*** - Kada "naslednik" završi izračunavanje funkcije koja mu je prosledjena, "roditeljskom" procesoru šalje ovu poruku. Kada "roditeljski" transpjuter dobije ovu poruku, on postavi na nulu vrednost niza **zauzet** za datog "naslednika".

5 ***KRAJRADA*** - Po završetku rada prvi transpjuter šalje ovu poruku svim ostalim (preko svojih direktnih "naslednika", pa dalje) transpjuterima.

6 ***NULP*** - Podatak koji se prenosi je nulti pokazivač (NULL pointer).

Promenljive:

ul i **izl** - Niz adresa 4 ulazna i 4 izlazna kanala

krajrada - Indikator da li je kraj rada ili ne?

stani - Ukoliko je vrednost 1 (poslata je poruka ***ZAHTEV***) stopira sve ostale poruke sve dok ne dobije poruku ***VREDNOST*** a zatim i izračunatu vrednost funkcije. Kada je vrednost 0, dozvoljava sve vrste poruka.

rac - Ako je vrednost promenljive jednaka 1, dati S-izraz (funkcija) je bio poslat na računanje nekom od "naslednika", pa je potrebno tražiti i rezultat (porukom ***ZAHTEV***). Ako je vrednost ove promenljive 0, dati transpjuter je sam izračunao vrednost.

zauzet - Koji su "naslednici" trenutno zauzeti?

pret - Redni broj "roditeljskog" transpjutera.

B.2.1.2. Prenos argumenata

Pošto paralelni C sadrži samo procedure za prenos celih brojeva ili znakova preko kanala, implementirane su procedure koje prenose argumente tipa **izraz, S-izraz, stanje** i **deffun** "naslednicima". Argumenti datog tipa su neophodni za izračunavanje vrednosti funkcije na transpjuteru "nasledniku". Ova celina sadrži procedure, koje vrše prenos datih struktura podataka:

primiizraz() - Prima strukturu tipa **izraz** sa datog kanala.

saljiizraz() - Šalje strukturu tipa **izraz** na dati kanal.

primistanje() - Prima strukturu tipa **stanje** sa datog kanala.

saljistanje() - Prosledjuje strukturu tipa **stanje** na dati kanal.

primilistuizr() - Prima sa datog kanala listu izraza.

saljilistuizr() - Šalje na dati kanal listu izraza.

primiime() - Prima redni broj imena funkcije ili promenljive sa datog kanala.

saljiime() - Šalje redni broj imena funkcije ili promenljive na dati kanal.

primilistuvr() - Prima listu vrednosti sa datog kanala.

saljelistuvr() - Šalje listu vrednosti na dati kanal.

primideffun() - Sa datog kanala prima definiciju funkcije.

saljideffun() - Na dati kanal salje definiciju funkcije.

B.2.1.3. Kontrolni deo

Ovo je najbitnija celina za paralelno izvršavanje. Procedure su sledeće:

odgovori() - Prima poruke sa ulaznih kanala i izvršava odgovarajuće akcije. U slučaju prvog transpjutera je jedina moguća akcija ***SLOBODNO***, a u slučaju ostalih transpjutera moguće su još i ***RACUNAJ***, ***ZAHTEV*** i ***KRAJRADA***.

radi() - Ispituje da li dati transpjuter ima naslednika, da li medju njima postoji neki slobodan, i da li je dati S-izraz korisnička funkcija. Ukoliko su svi uslovi zadovoljeni šalje datu funkciju na izračunavanje prvom slobodnom svom "nasledniku", inače joj sam izračunava vrednost. Promenljive **rac** i **s** su pokazivači na vrednosti koji kazuju da li je i kojim kanalom slana funkcija na izračunavanje, da bi znali da li treba i sa kog kanala tražiti njenu vrednost.

uzmi() - Šalje zahtev za izračunatom vrednošću, zatim blokira sve ulazne kanale, osim datog, sve dok ne dobije datu vrednost.

izraclistu() - Izračunava listu izraza (svaki od njih) i beleži u listi vrednosti. Jedina promena u odnosu na DOS verziju je u tome što ne poziva **izracunaj()** nego **radi()**, i što u slučaju prosledjivanja na izračunavanje (**rac** = **1**) poziva na kraju i funkciju **uzmi()** za prijem rezultata od "naslednika" koji je izvršio računanje.

izrac() - Izračunava vrednost funkcije (**izracunaj()**) a zatim prosledjuje poruku ***SLOBODAN*** "roditeljskom" transpjuteru.

uradi() - Ostaje nepromenjen

glavni() - Preimenovan **main()** iz PC verzije, uz dodatno slanje poruke za kraj rada na kraju procedure (izvršavanja programa).

main() - Inicijalizuje potrebne promenljive, a zatim poziva **glavni()**.

B.2.2. Deo koji se izrvršava na ostalim transpjuterima

Deo programa koji se izvršava na ostalim transpjuterima je izmenjen u odnosu na deo koji se izvršava na prvom transpjuteru. Celine koje su zajedničke za oba dela su:

❶ Deklaracije,

❷ Izračunavanje vrednosti izraza,

❸ Glavni program, i

❹ Prenos argumenata

Celina Ulazni podaci i definicija korisničkih funkcija je izbačena kao nepotrebna. Razlog je nemogućnost ostalih transpjutera za ulazno-izlaznim operacijama.

U celini Podrška osnovnim tipovima podataka ostale su samo procedure: **naplistaim(), naplistavr(), napstanje(), duzlistaim()** i **duzlistavr().**

Celina Operacije imenima funkcija sadrži samo procedure: **uzmideffun()** i **postaviim()**

Celina Postavljanje parametara je zadržala sve procedurte osim **prazst()**.

U celini Manipulacija S-izrazima nedostaje samo **stamsizr()**.

Celina Prenos argumenata sadrži sledeće nove procedure:

stavisizr() - Stavlja S-izraz na stek za komunikaciju. Dati S-izraz je izračunata vrednost funkcije, koja je dobijena na izračunavanje od "roditeljskog" transpjutera.

uzmisizr() - Uzima rezultat sa steka za komunikaciju. Jedna od procedura **odgporuke()** ili **odgovori()** , zatim šalje dati rezultat roditeljskom transpjuteru.

U Kontrolnom delu su u proceduri **odgovori()** dodati novi slučajevi (***RACUNAJ, ZAHTEV*** i ***KRAJRADA***). Promenjena je i procedura **main()** pa je umesto poziva **glavni()** poziv procedure **odgovori().**

DODATAK C Izvorni kod LISP interpretera

C.1. Kod koji se izvršava na prvom transpjuteru

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "conc.h"


/* DEFINICIJE */


#define BROJ_NASL 2
#define BROJ_TRANSP 17
#define IMA_NASL 8
#define RACUNAJ 1
#define ZAHTEV 2
#define VREDNOST 3
#define SLOBODNO 4
#define KRAJRADA 5
#define PORUKA 7
#define NULP -2
#define MAXST   10000
#define DUZPORUK 20
#define velmem 5000
#define duzimena 20 /* maksimalna duzina imena */
#define maximena 200  /*maksimalan broj razlicitih
imena*/
#define maxulaz 2500  /* maksimalna duzina ulaza*/
#define prompt "-> "
#define prompt2 "> "
#define komentar ';'
#define tabkod 9        /*  ascii kod */
#define argstekvel 10000
typedef int  velimena;
typedef char imestr[duzimena+1];
typedef int  number;
typedef int  ime; /* a ime je indeks u nizimena */
typedef enum{ifop, condop, whileop, setop, setqop,
                beginop, plusop, minusop,  timesop,
                divop, eqop,ltop,       gtop, consop,
                carop, cdrop, numberpop, symbolpop,
                listpop, nullpop, printop } korop;
#define listaizr struct listaizrstr *
typedef int  vrop;
typedef int  konop;
#define sizr struct sizrstr *
typedef enum {nulasizr,brsizr,simsizr,lsizr} tipsizr;
#define listavr struct listavrstr *
typedef enum {brizr,promizr,funizr, lisizr} tipizr;
typedef struct izrazstr
{
   tipizr itip;  /* tip izraza */
   union
   {
      struct { sizr siz; } sbrizr;
      struct { ime promen; int ofset; } spromizr;
      struct { ime oper; listaizr argum; } sfunizr;
      struct { listaizr argum; } slizr;
   } v;
} izrazstr;

#undef listavr
typedef struct listavrstr *listavr;
typedef struct sizrstr
{
   int refbr;
   tipsizr tsizr;
   union
    {
      struct { int gg; } snulasizr;
      struct { int brvr; } sbrsizr;
      struct { ime simvr; } ssimsizr;
      struct { sizr pocvr; sizr ostvr;} slsizr;
   } v;
} sizrstr;

typedef struct listavrstr
{
   sizr pocetak;
   listavr ostatak;
} listavrstr;

typedef izrazstr *izraz;
#define stanje struct stanjestr *
#define listaim struct listaimstr *
#define deffun struct deffunstr *
typedef struct listaizrstr
{
   izraz pocetak;
   listaizr ostatak;
```

```
} listaizrstr;

#undef deffun
typedef struct deffunstr *deffun;
typedef struct listaimstr
{
  ime pocetak;
  listaim ostatak;
} listaimstr;

typedef struct stanjestr
{
  listaim stprom;
  listavr stvred;
} stanjestr;

typedef struct deffunstr
{
  ime imefun;
  listaim formarg;
  izraz telofun;
  deffun sleddeffun;
} deffunstr;


/* DEKLARACIJE PROMENLJIVIH */


Channel *ul[4], *izl[4];
int krajrada, stani, rac;
int zauzet[BROJ_NASL+1];
sizr st[MAXST];
int velst,i;
FILE *ulaz, *izlaz;
char imeulazne[10];
int pret[20];
deffun svedeffun;
stanje ukstanje;
izraz tekizr;
sizr nulavr;
sizr tacnavr;
char ulazpod[maxulaz];
int  duzulaza, poz;
imestr nizimena[200];
ime brimena, brugradjenih;
sizr argstek[argstekvel];
int vrhargstek;
sizr slobsizr;
int vr1,vr2,s,m,sek,ms,miks;


/* UPRAVLJANJE MEMORIJOM */


void uradi(void);

/* initmem - alocira memoriju za sve sizrstr strukture
*/
void initmem(void)
{  int i;
   sizr s;

  slobsizr = NULL;
    for (i = 1; i <= velmem; i++)
      {
      s = slobsizr;
      slobsizr = malloc(sizeof(*slobsizr));
      slobsizr->v.slsizr.pocvr = s;
      }
} /* initmem */

/* alocsizr - alocira sizrstr strukturu iz liste slobodnih
sizraza */
sizr alocsizr(tipsizr t)
{  sizr s;
   if (slobsizr = = NULL)
      {
        printf("Nema slobodne memorije\n");
        uradi();
      }
      s = slobsizr;
      s->tsizr = t;
      s->refbr = 0;
      slobsizr = slobsizr->v.slsizr.pocvr;
      return s;
} /* alocsizr */

/* vratisizr - vraca sizrstr u listu slobodnix sizraza  */
void vratisizr(sizr s)
{
      s->v.slsizr.pocvr = slobsizr;
      slobsizr = s;
} /* vratisizr */

/* povrefbr - povecava refbr polje argumenta */
void povrefbr(sizr s)
{
      s->refbr = s->refbr + 1;
} /* povrefbr */

/* smarefbr - smanjuje refbr polje argumenta */
void smarefbr(sizr s)
{
  s->refbr = s->refbr - 1;
  if (s->refbr = = 0)
  {
    if (s->tsizr = = lsizr)
    {
      smarefbr(s->v.slsizr.pocvr);
      smarefbr(s->v.slsizr.ostvr);
    }
    vratisizr(s);
  }
} /* smarefbr */

/*oslrefbr - smanjuje refbr u svim sizrazima
izvan iz*/
void oslrefbr(izraz iz)
```

```
{ listaizr argl;
  switch (iz->itip)
  {
    case brizr:   smarefbr(iz->v.sbrizr.siz);
      break;

    case promizr: ;
      break;

    case funizr:
      argl = iz->v.sfunizr.argum;
      while (argl != NULL)
      {
        oslrefbr(argl->pocetak);
        argl = argl->ostatak;
      }
      break;
  }
} /* oslrefbr */

/* iniargstek - inicijalizuje argument stek     */
void iniargstek(void)
{
  vrhargstek = 1;
} /* iniargstek */

/* staviarg - stavlja jedan s-izraz na argument stek */
void staviarg(sizr s)
{
  if (vrhargstek > argstekvel)
    {
      printf("prekoracenje steka\n");
      vrhargstek = 1;
      uradi();
    }
  argstek[vrhargstek-1] = s;
  vrhargstek = vrhargstek + 1;
  povrefbr(s);
} /* staviarg */

/* uzmivrharg - vraca vrh argument steka */
sizr uzmivrharg(void)
{
return argstek[vrhargstek - 2];
} /* uzmivrharg */

/* uzmidovrharg - vraca prvi s-izraz ispod vrha */
sizr uzmidovrharg(void)
{
  if (vrhargstek > 2) return argstek[vrhargstek - 3];
} /* uzmidovrharg */

/* izbaciarg - izbacuje dati broj s-izraza sa argument
steka */
void izbaciarg(int l)
{ int i;
  for (i = 1; i <= l; i++)
    {
      smarefbr(argstek[vrhargstek - 2]);
      vrhargstek = vrhargstek - 1;
    }
} /* izbaciarg */


/* PODRSKA OSNOVNIM TIPOVIMA
PODATAKA */


/* napvrizr - vraca izraz tipa brizr */
izraz napvrizr(sizr s)
{ izraz iz;
  iz = malloc(sizeof(*iz));
  iz->itip = brizr;
  iz->v.sbrizr.siz = s;
  povrefbr(s);
  return iz;
} /* napvrizr */

/* napprizr - vraca izraz tipa promizr */
izraz napprizr(ime im)
{ izraz iz;
  iz = malloc(sizeof(*iz));
  iz->itip = promizr;
  iz->v.spromizr.promen = im;
  iz->v.spromizr.ofset = 0;
  return iz;
} /* napprizr */

/* napfunizr - vraca izraz tipa funizr operacije op i sa
argumentima lizr */
izraz napfunizr(ime op, listaizr lizr)
{ izraz iz;
  iz = malloc(sizeof(*iz));
  iz->itip = funizr;
  iz->v.sfunizr.oper = op;
  iz->v.sfunizr.argum = lizr;
  return iz;
} /* napfunizr */

/* naplistizr - vraca izraz of tipa funizr sa
argumentom lizr */
izraz naplistizr(listaizr lizr)
{ izraz iz;
  iz = malloc(sizeof(*iz));
  iz->itip = lisizr;
  iz->v.slizr.argum = lizr;
  return iz;
} /* napfunizr */

/* naplistaizr - vraca listu izraza sa pocetkom iz i
ostatkom lizr */
listaizr naplistaizr(izraz iz, listaizr lizr)
{ listaizr novalizr;
  novalizr = malloc(sizeof(*novalizr));
  novalizr->pocetak = iz;
  novalizr->ostatak = lizr;
  return novalizr;
} /* naplistaizr */
```

```
/* naplistaim - vraca listu imena sa pocetkom n i
ostatkom limena */
listaim naplistaim(ime im, listaim limena)
{ listaim novalimena;
   novalimena = malloc(sizeof(*novalimena));
   novalimena->pocetak = im;
   novalimena->ostatak = limena;
   return novalimena;
} /* naplistaim */

/* naplistavr - vraca listu vrednosti sa pocetkom n i
ostatkom lvred */
listavr naplistavr(sizr n, listavr lvred)
{ listavr novalvred;
   novalvred = malloc(sizeof(*novalvred));
   novalvred->pocetak = n;
   novalvred->ostatak = lvred;
   return novalvred;
} /* naplistavr */

/* napstanje - vraca stanje sa promenljivima limena i
vrednostima lvred */
stanje napstanje(listaim limena, listavr lvred)
{ stanje rho;
   rho = malloc(sizeof(*rho));
   rho->stprom = limena;
   rho->stvred = lvred;
   return rho;
} /* napstanje */

/* duzlistavr - vraca duzinu liste vrednosti lvred */
int duzlistavr(listavr lvred)
{ int i;
   i = 0;
   while (lvred != NULL)
   {
      i = i + 1;
      lvred = lvred->ostatak;
   }
   return i;
} /* duzlistavr */

/* duzlistaim - vraca duzinu liste imena limena */
int duzlistaim(listaim limena)
{ int i;
   i = 0;
   while (limena != NULL)
      {
         i = i + 1;
         limena = limena->ostatak;
      }
   return i;
} /* duzlistaim */


/*  OPERACIJE NAD IMENIMA FUNKCIJA      */
```

```
/*uzmideffun - uzima definiciju funkcije fime iz
svedeffun */
deffun uzmideffun(ime fime)
{ deffun f; int nasao;
   nasao = 0;
   f = svedeffun;
   while ((f != NULL) && !nasao)
      if (f->imefun = = fime) nasao = 1;
      else f = f->sleddeffun;
   return f;
} /* uzmideffun */

/*novadeffun - dodaje novu definiciju funkcije
imenom fime parametrima limena,
i telom funkcije iz */
void novadeffun(ime fime, listaim limena, izraz iz)
{ deffun f;
   f = uzmideffun(fime);
   if (f = = NULL)
   {
      f = malloc(sizeof(*f));
      f->sleddeffun = svedeffun;
      svedeffun = f;
   }
   f->imefun = fime;
   f->formarg = limena;
   f->telofun = iz;
} /* novadeffun */

/* postaviim - postavlja sva definisana imena u
nizimena */
void postaviim(void)
{ int i;
   strcpy(ulazpod,".                    ");
   svedeffun = NULL;
   i = 1;
   strcpy(nizimena[i-1],"if                ");  i = i + 1;
   strcpy(nizimena[i-1],"cond              ");  i = i + 1;
   strcpy(nizimena[i-1],"while             ");  i = i + 1;
   strcpy(nizimena[i-1],"set               ");  i = i + 1;
   strcpy(nizimena[i-1],"setq              ");  i = i + 1;
   strcpy(nizimena[i-1],"begin           ");  i = i + 1;
   strcpy(nizimena[i-1],"+                 ");  i = i + 1;
   strcpy(nizimena[i-1],"-               ");  i = i + 1;
   strcpy(nizimena[i-1],"*               ");  i = i + 1;
   strcpy(nizimena[i-1],"/                 ");  i = i + 1;
   strcpy(nizimena[i-1],"=                 ");  i = i + 1;
   strcpy(nizimena[i-1],"<                 ");  i = i + 1;
   strcpy(nizimena[i-1],">                 ");  i = i + 1;
   strcpy(nizimena[i-1],"cons              ");  i = i + 1;
   strcpy(nizimena[i-1],"car             ");  i = i + 1;
   strcpy(nizimena[i-1],"cdr             ");  i = i + 1;
   strcpy(nizimena[i-1],"number?         ");  i = i + 1;
   strcpy(nizimena[i-1],"symbol?         ");  i = i + 1;
   strcpy(nizimena[i-1],"list?           ");  i = i + 1;
   strcpy(nizimena[i-1],"null?           ");  i = i + 1;
   strcpy(nizimena[i-1],"print             ");  i = i + 1;
   strcpy(nizimena[i-1],"T                 ");
   brimena = i;
```

```
   brugradjenih = i;
} /* postaviim */

/* novoime - dodaje novo ime u nizimena */
ime novoime(char * im)
{ int i; int nasao;
   i = 1;
   nasao = 0;
   im[20] = '\x0';
   while ((i <= brimena) && !nasao)
      if (strcmp(im,nizimena[i-1]) = = 0) nasao = 1;
      else i = i + 1;
   if (!nasao)
   {
      if (i > maximena)
      {
         printf("nema vise mesta za imena\n");
         uradi();
      }
      brimena = i;
      strcpy(nizimena[i-1],im);
   }
   return i;
} /* novoime */

/* stamime - stampa ime im */
void stamime(ime im)
{ int i;
   i = 0;
   while (i <= duzimena)
   if (nizimena[im-1][i] != ' ')
   {
      printf("%c",nizimena[im-1][i]);
      i = i + 1;
   }
   else i = duzimena + 1;
} /* stamime */

/* nalaziop - nalazi za ime odgovarajuci korop */
korop nalaziop(ime oper)
{ korop op;  int i;
   op = ifop;
   for (i = 1; i <= oper - 1; i++) op++;
   return op;
 } /* nalaziop */


/* ULAZ PODATAKA I DEFINICIJA KORISNICKI
DEFINISANIH FUNKCIJA */


void postizrprom(izraz iz, listaim limena);
int dalije_broj(int poz);
int izbpraz(int p);
ime izdvime(void);
listaim izdvlistim(void);
izraz izdvizr(void);
sizr izdvlistsizr(sizr s);
sizr izdvsizr(void);

/* ofsetprom - nalazi promenljivu */
int ofsetprom(ime im, listaim limena)
{ int i;
   int nasao;

   i = 1;
   nasao = 0;
   while ((limena != NULL) && !nasao)
      if (im = = limena->pocetak) nasao = 1;
      else
      {
         i = i + 1;
         limena = limena->ostatak;
      }
      if (!nasao) i = 0;
      return i;
} /* ofsetprom */

/* postlizrprom - primenjuje postizrprom na svaki
izraz u listi izraza lizr */
void postlizrprom(listaizr lizr, listaim limena)
{
   while (lizr != NULL)
   {
      postizrprom(lizr->pocetak, limena);
      lizr = lizr->ostatak;
   }
} /* postlizrprom */

/* postizrprom - postavlja ofset promenljivima u listi
imena limena u   izrazu iz */
void postizrprom(izraz iz, listaim limena)
{
   switch (iz->itip)
   {
      case brizr:
         break;
      case promizr: iz->v.spromizr.ofset =
            ofsetprom(iz->v.spromizr.promen,limena);
         break;

      case funizr: postlizrprom(iz->v.sfunizr.argum,
limena);
         break;
   }
} /* postizrprom */

/* izdvfunkc - izdvaja definiciju funkcije iz ulaznog
niza */
ime izdvfunkc(void)
{  ime fime;
   listaim limena;
   izraz iz;

   poz = izbpraz(poz + 1);
   poz = izbpraz(poz + 6);
   fime = izdvime();
   poz = izbpraz(poz + 1);
```

```
  limena = izdvlistim();
  iz = izdvizr();
  poz = izbpraz(poz + 1);
  novadeffun(fime,limena,iz);
  postizrprom(iz, limena);
  return fime;
} /* izdvfunkc */

/* dalije_broj - ispituje da li se u ulaznom nizu
pojavio broj ili nesto drugo */
int dalije_broj(int poz)
{
  return (ulazpod[poz-1] = = '\'') || dalije_broj(poz);
}/* dalije_broj */

/* izdvceob - izdvaja ceo broj iz ulaznog niza */
sizr izdvceob(sizr s)
{ int sum, sign;
  s = alocsizr(brsizr);
  sum = 0;
  sign = 1;
  if (ulazpod[poz-1] = = '-')
  {
    sign = -1;
    poz = poz + 1;
  }
  while (ulazpod[poz-1]>='0' &&
          ulazpod[poz-1]<='9')
  {
    sum = 10 * sum + (ulazpod[poz-1] - '0');
    poz = poz + 1;
  }
  s->v.sbrsizr.brvr = sum * sign;
  poz = izbpraz(poz);
  return s;
} /* izdvceob */

/* izdvsimb - izdvaja simbol iz ulaznog niza */
sizr izdvsimb(sizr s)
{
  s = alocsizr(simsizr);
  s->v.ssimsizr.simvr = izdvime();
  return s;
} /* izdvsimb */

/* izdvlistsizr - izdvaja listu izraza */
sizr izdvlistsizr(sizr s)
{ sizr poc;
  sizr ost;

  if (ulazpod[poz-1] = = ')')
  {
    poz = izbpraz(poz + 1);
    return alocsizr(nulasizr);
  }
  else
  {
    poc = izdvsizr();
    ost = izdvlistsizr(s);
    s = alocsizr(lsizr);
    s->v.slsizr.pocvr = poc;
    povrefbr(poc);
    s->v.slsizr.ostvr = ost;
    povrefbr(ost);
    return s;
  }
} /* izdvlistsizr */

/* izdvsizr - izdvaja s-izraz iz ulaznog niza */
sizr izdvsizr(void)
{ sizr s;

  if (dalije_broj(poz)) return izdvceob(s);
  else if (ulazpod[poz-1] = = '(')
       {
         poz = izbpraz(poz + 1);
         return izdvlistsizr(s);
       }
       else return izdvsimb(s);
} /* izdvsizr */

/* izdvvr - izdvaja vrednost iz ulaznog niza */
sizr izdvvr(void)
{
  if (ulazpod[poz-1] = = '\'') poz = poz + 1;
  return izdvsizr();
} /* izdvvr */

/* dalije_delim - ispituje da li je delimiter */
int dalije_delim(char c)
{
  return (c=='(' || c==')' || c==' ' || c==komentar);
} /* dalije_delim */

/* izbpraz - izbacuje nepotrebne blankove iz ulaznog
niza */
int izbpraz(int p)
{
  while (ulazpod[p-1] = = ' ') p = p + 1;
  return p;
} /* izbpraz */

/* ististr - da li je rec im ista kao i na ulazu */
int ististr(int s, velimena leng,  char * im)
{ int match;
  int i;

  match = 1;
  i = 1;
  while (match && (i <= leng))
  {
    if (ulazpod[s-1] != im[i-1]) match = 0;
    i = i + 1;
    s = s + 1;
  }
  if (!dalije_delim(ulazpod[s-1])) match = 0;
  return match;
} /* ististr */
```

```
/* sledslovo - ucitaj sledece slovo sa ulaza */
void sledslovo(char * c)
{
   fscanf(ulaz,"%c",c);
   if (*c = = tabkod) *c = ' ';
   else if (*c = = komentar)
         {
            while (*c!='\n') fscanf(ulaz,"%c",c);
            *c = ' ';
         }
} /* sledslovo */

/* ucitblok - ucitava blok ulaznih podataka */
void ucitblok(void)
{ int parencnt;
   char c;

   parencnt = 1;
   do {
         if (c=='\n') printf("%s",prompt2);
         sledslovo(&c);
         poz = poz + 1;
         if (poz = = maxulaz)
            printf("ulazni podaci preveliki\n");
         if (poz = = maxulaz)  uradi();
         if (c >= 32) ulazpod[poz-1] = c;
         else ulazpod[poz-1] = ' ';
         if (c = = '(') parencnt = parencnt + 1;
         if (c = = ')') parencnt = parencnt - 1;
      } while (parencnt != 0);
} /* ucitblok */

/* ucitulaz - ucitava ulaz */
void ucitulaz(void)
{ char c;

   printf("%s",prompt);
   poz = 0;
   do {
         poz = poz + 1;
         if (poz = = maxulaz-1)
         {
            fclose(ulaz);
            ulaz = stdin;
            poz = 1;
         }
         sledslovo(&c);
         if (c >= 32) ulazpod[poz-1] = c;
         else ulazpod[poz-1] = ' ';
         if (ulazpod[poz-1] = = '(') ucitblok();
      }  while (c!='\n');
      duzulaza = poz;
      ulazpod[poz] = komentar;
} /* ucitulaz */

/* ucitaj - ucitava ceo ulazni niz */
void ucitaj(void)
{ char ime[12];
   int i;

   do {
         ucitulaz();
         poz = izbpraz(1);
      } while (!(poz <= duzulaza));
   if (ististr(izbpraz(poz+1),4,"load               "))
   {
      strcpy(ime,"          ");
      for (i=0; i<12 && ulazpod[poz+5+i]!=')'); i++)
         ime[i] = ulazpod[poz+5+i];
      ulaz = fopen(ime,"rt");
      ucitaj();
   }
} /* ucitaj */

/* izdvime - izdvaja (vec korisceno) ime iz ulaznog
niza */
ime izdvime(void)
{ imestr im;
   velimena le, leng;

   leng = 0;
   strcpy(im,"                ");
   while ((poz <= duzulaza) &&
         !dalije_delim(ulazpod[poz-1]))
   {
      if (leng = = duzimena)
      {
         printf("predugo ime, pocetak je: %s\n",im);
         uradi();
      }
      leng = leng + 1;
      im[leng-1] = ulazpod[poz-1];
      poz = poz + 1;
   }
   if (leng = = 0)
   {
      printf("greska: potrebno ime, umesto:",
            "%c\n",ulazpod[poz-1]); uradi();
   }
   for (le = leng + 1; le <= duzimena; le++)
      im[le-1] = ' ';
   leng = le;
   poz = izbpraz(poz);
   return novoime(im);
} /* izdvime */

/* dalije_cifra - ispituje da li je dato slovo cifra */
int dalije_cifra(int poz)
{
   if (ulazpod[poz-1] < '0' || ulazpod[poz-1] > '9')
      return 0;
   else
   {
      while (ulazpod[poz-1]>='0' &&
                  ulazpod[poz-1]<='9')
         poz = poz+1;
      if (!dalije_delim(ulazpod[poz-1])) return 0;
```

```
    return 1;
  }
} /* dalije_cifra */

/* dalije_broj - da li je broj */
int dalije_broj(int poz)
{
  return dalije_cifra(poz) || ((ulazpod[poz-1] = = '-')
    &&  dalije_cifra(poz + 1));
} /* dalije_broj */

listaizr izdvlistizr(void);

/* izdvizr - izdvaja izraz iz ulaznog niza */
izraz izdvizr(void)
{ ime im;
  listaizr lizr;

  if (ulazpod[poz-1] = = '(')
  {
    poz = izbpraz(poz + 1);
    if (ulazpod[poz-1] = = '(')
    {
      lizr = izdvlistizr();
      return naplistizr(lizr);
    }
    else
    {
      im = izdvime();
      lizr = izdvlistizr();
      return napfunizr(im,lizr);
    }
  }
  else if (dalije_broj(poz)) /* brizr */
        return napvrizr(izdvvr());
      else /* promizr */
        return napprizr(izdvime());
} /* izdvizr */

/* izdvlistizr - izdvaja listu izraza */
listaizr izdvlistizr(void)
{ izraz iz;
  listaizr lizr;

  if (ulazpod[poz-1] = = ')')
  {
    poz = izbpraz(poz + 1);
    return NULL;
  }
  else
  {
    iz = izdvizr();
    lizr = izdvlistizr();
    return naplistaizr(iz,lizr);
  }
} /* izdvlistizr */

/* izdvlistim - izdvaja listu imena */
listaim izdvlistim(void)
{ ime im;
  listaim limena;

  if (ulazpod[poz-1] = = ')')
  {
    poz = izbpraz(poz + 1);
    return NULL;
  }
  else
  {
    im = izdvime();
    limena = izdvlistim();
    return naplistaim(im,limena);
  }
} /* izdvlistim */


/* POSTAVLJANJE PARAMETARA */


/* prazst - vraca prazno stanje */
stanje prazst(void)
{
    return napstanje(NULL,NULL);
} /* prazst */

/* postprom - postavlja promenljivoj im vrednost n u
stanju rho */
void postprom(ime im, sizr s, stanje rho)
{
  povrefbr(s);
  rho->stprom = naplistaim(im,rho->stprom);
  rho->stvred = naplistavr(s,rho->stvred);
} /* postprom */

/* nadjiprom - nalazi promenljivu im u stanju rho */
listavr nadjiprom(ime im, stanje rho)
{ listaim limena;
  listavr lvred;
  int nasao;

  nasao = 0;
  limena = rho->stprom;
  lvred = rho->stvred;
  while ((limena != NULL) && !nasao)
    if (limena->pocetak = = im) nasao = 1;
    else
    {
      limena = limena->ostatak;
      lvred = lvred->ostatak;
    }
  return lvred;
} /* nadjiprom */

/* dodprom - dodeli promenljivoj im vrednost n u
stanju rho */
void dodprom(ime im, sizr s, stanje rho)
{ listavr varloc;
```

```
   povrefbr(s);
   varloc = nadjiprom(im,rho);
   smarefbr(varloc->pocetak);
   varloc->pocetak = s;
} /* dodprom */

/* vrprom - vraca vrednost promenljive im u stanju
rho */
sizr vrprom(ime im, stanje rho)
{ listavr lvred;

   lvred = nadjiprom(im,rho);
   return lvred->pocetak;
} /* vrprom */

/* dalije_prom - da li je dato ime promenljiva ili ne u
stanju rho */
int dalije_prom(ime im, stanje rho)
{
   return nadjiprom(im,rho) != NULL;
} /* dalije_prom */


/* MANIPULACIJA S-IZRAZIMA */


/* stamsizr - stampa s-izraz s */
void stamsizr(sizr s)
{ sizr s1;

   switch (s->tsizr)
   {
      case nulasizr: printf("()");
         break;

      case brsizr: printf("%1d",s->v.sbrsizr.brvr);
         break;

      case simsizr: stamime(s->v.ssimsizr.simvr);
         break;

      case lsizr:
         printf("(");
         stamsizr(s->v.slsizr.pocvr);
         s1 = s->v.slsizr.ostvr;
         while (s1->tsizr = = lsizr)
         {
            printf(" ");
            stamsizr(s1->v.slsizr.pocvr);
            s1 = s1->v.slsizr.ostvr;
         }
         printf(")");
         break;
      }
} /* stamsizr */

/* dalije_tacan - vraca 1 ako nije nulti s-izraz */
int dalije_tacan(sizr s)
{
   return s->tsizr != nulasizr;
} /* dalije_tacan */

/* primoper - primeni aritmeticku operaciju */
void primoper(int n1, int n2, vrop op, sizr *rezultat)
{ sizr pom;

   pom = alocsizr(brsizr);
   switch (op)
   {
      case plusop: pom->v.sbrsizr.brvr = n1 + n2;
         break;

      case minusop: pom->v.sbrsizr.brvr = n1 - n2;
         break;

      case timesop: pom->v.sbrsizr.brvr = n1 * n2;
         break;

      case divop: pom->v.sbrsizr.brvr = n1 / n2;
         break;
   }
   *rezultat = pom;
} /* primoper */

/* primrel - primeni relaciju na argumente */
void primrel(int n1, int n2, vrop op, sizr *rezultat)
{
   switch (op)
   {
      case ltop: if (n1 < n2) *rezultat = tacnavr;
         break;
      case gtop: if (n1 > n2) *rezultat = tacnavr;
         break;
   }
} /* primrel */

/* brargum - nalazi broj argumenata operacije */
int brargum(vrop op)
{
   if (op>=plusop && op<=consop) return 2;
   else return 1;
} /* brargum */

/* primvrop - primenjuje datu operaciju */
void primvrop(vrop op)
{ sizr rezultat;
   sizr s1;
   sizr s2;

   rezultat = nulavr;
   s1 = uzmivrharg();
   if (brargum(op) = = 2)
   {
      s2 = s1;
      s1 = uzmidovrharg();
   }
   if ((op>=plusop && op<=divop) ||
       (op>=ltop && op<=gtop))
```

```
    if ((s1->tsizr = = brsizr) &&
       (s2->tsizr = = brsizr))
       if (op>=plusop && op<=divop)
          primoper(s1->v.sbrsizr.brvr,
                   s2->v.sbrsizr.brvr,
  op,&rezultat);
       else primrel(s1->v.sbrsizr.brvr,
                   s2->v.sbrsizr.brvr,    op,
&rezultat);
    else
    {
       printf("argumenti nisu aritmeticke vrednosti ");
       stamime(op + 1);
       printf("\n");
       uradi();
    }
  else
    switch (op)
    {
       case eqop:
          if ((s1->tsizr = = nulasizr) &&
          (s2->tsizr = = nulasizr))
             rezultat = tacnavr;
          else if ((s1->tsizr = = brsizr)
               && (s2->tsizr = = brsizr) &&
               (s1->v.sbrsizr.brvr = =
               s2->v.sbrsizr.brvr))
                  rezultat = tacnavr;
               else if ((s1->tsizr = = simsizr) &&
                    (s2->tsizr = = simsizr) &&
                    (s1->v.ssimsizr.simvr = =
                    s2->v.ssimsizr.simvr))
                       rezultat = tacnavr;
          break;

       case consop:
          rezultat = alocsizr(lsizr);
          rezultat->v.slsizr.pocvr = s1;
          povrefbr(s1);
          rezultat->v.slsizr.ostvr = s2;
          povrefbr(s2);
          break;

       case carop:
          if (s1->tsizr != lsizr)
          {
             printf("greska poc primenjen na ne listu");
             stamsizr(s1);
             printf("\n");
          }
          else rezultat = s1->v.slsizr.pocvr;
          break;

       case cdrop:
          if (s1->tsizr != lsizr)
          {
             printf("greska ost primenjen na ne-listu);
             stamsizr(s1);
             printf("\n");
          }
          else rezultat = s1->v.slsizr.ostvr;
          break;

       case numberpop:
          if (s1->tsizr = = brsizr) rezultat = tacnavr;
          break;

       case symbolpop:
          if (s1->tsizr = = simsizr) rezultat = tacnavr;
             break;

       case listpop:
          if (s1->tsizr = = lsizr) rezultat = tacnavr;
          break;

       case nullpop:
          if (s1->tsizr = = nulasizr) rezultat = tacnavr;
          break;

          case printop:
             stamsizr(s1);
             printf("\n");
             rezultat = s1;
             break;
       }
       izbaciarg(brargum(op));
       staviarg(rezultat);
} /* primvrop */


/* IZRACUNAVANJE VREDNOSTI IZRAZA */


void izracunaj(izraz iz, int ar);
void izraclistu(listaizr lizr, int ar);

/* primkorop - primenjuje korisnicki definisanu
funkciju */
void primkorop(ime im, int newar)
{  deffun f;
   sizr s;

   f = uzmideffun(im);
   if (f = = NULL)
   {
      printf("nedefinisana funkcija: ");
      stamime(im);
      printf("\n");
      uradi();
   }
   else
   {
      izracunaj(f->telofun,newar);
      s = uzmivrharg();
      vrhargstek = vrhargstek - 1;
      izbaciarg(duzlistaim(f->formarg));
      argstek[vrhargstek-1] = s;
      vrhargstek = vrhargstek + 1;
```

```
  }
 } /* primkorop */

/* primkonop - primenjuje kontrolnu operaciju */
void primkonop(konop op, listaizr argum, int ar)
{ sizr s;

  switch (op)
  {
    case ifop: izracunaj(argum->pocetak,ar);
      s = uzmivrharg();
      if (dalije_tacan(s))
      {
        izbaciarg(1);
        izracunaj(argum->ostatak->pocetak,ar);
      }
      else
      {
        izbaciarg(1);
        izracunaj(argum->ostatak->ostatak
                  ->pocetak,ar);
      }
      break;

    case condop:
      izracunaj(argum->pocetak->v.slizr.argum-
>pocetak,
                ar);
      s = uzmivrharg();
      while((argum->ostatak != NULL) &&
            !dalije_tacan(s))
      {
        izracunaj(argum->pocetak->v.slizr.argum->
                  pocetak, ar);
        s = uzmivrharg();
        if (!dalije_tacan(s))
          argum = argum->ostatak;
      }
      if (argum->ostatak != NULL)
      {
        izracunaj(argum->pocetak->v.slizr.argum
                  ->ostatak->pocetak, ar);
        s = uzmivrharg();
      }
      else
      {
        izracunaj(argum->pocetak->v.slizr.argum
                  ->pocetak,    ar);
        s = uzmivrharg();
        if (dalije_tacan(s))
        {
          izracunaj(argum->pocetak->v.slizr.argum
                  ->ostatak->pocetak, ar);
          s = uzmivrharg();
        }
        staviarg(s);
      }
      break;

case whileop:
  staviarg(nulavr);
  izracunaj(argum->pocetak,ar);
  s = uzmivrharg();
  while (dalije_tacan(s))
  {
    izbaciarg(2);
    izracunaj(argum->ostatak->pocetak,ar);
    izracunaj(argum->pocetak,ar);
    s = uzmivrharg();
  }
  izbaciarg(1);
  break;

case setop:
  izracunaj(argum->ostatak->pocetak,ar);
  s = uzmivrharg();
  if (argum->pocetak->v.spromizr.ofset > 0)
  {
    povrefbr(s);
    smarefbr(argstek[ar +
    argum->pocetak->v.spromizr.ofset - 2]);
    argstek[ar + argum->pocetak
            ->v.spromizr.ofset - 2] = s;
  }
  else if (dalije_prom(argum->pocetak
             ->v.spromizr.promen,ukstanje))
         dodprom(argum->pocetak
           ->v.spromizr.promen,s,ukstanje);
       else
         postprom(argum->pocetak
                  ->v.spromizr.promen,s,ukstanje);
  break;

case setqop:
  izracunaj(argum->ostatak->pocetak,ar);
  s = uzmivrharg();
  if (argum->pocetak->v.spromizr.ofset > 0)
  {
    povrefbr(s);
    smarefbr(argstek[ar + argum->pocetak
                  ->v.spromizr.ofset - 2]);
    argstek[ar + argum->pocetak
                  ->v.spromizr.ofset - 2] = s;
  }
  else if (dalije_prom(argum->pocetak->
       v.spromizr.promen, ukstanje))
         dodprom(argum->pocetak
                  ->v.spromizr.promen,s,ukstanje);
       else
         postprom(argum->pocetak
                  ->v.spromizr.promen,s,ukstanje);
  break;

  case beginop:
    while (argum->ostatak != NULL)
    {
      izracunaj(argum->pocetak,ar);
      izbaciarg(1);
```

```
            argum = argum->ostatak;
          }
          izracunaj(argum->pocetak,ar);
          break;
      }
} /* primkonop */

/* izracunaj - izracunava vrednost izraza */
void izracunaj(izraz iz, int ar)
{ sizr s;
  korop op;
  int newar;

  switch (iz->itip)
  {
    case brizr: staviarg(iz->v.sbrizr.siz);
      break;

    case promizr:
      if (iz->v.spromizr.ofset > 0)
        s = argstek[ar + iz->v.spromizr.ofset - 2];
      else if (dalije_prom(iz->v.spromizr.promen,
          ukstanje))
            s = vrprom(iz->v.spromizr.promen,
                       ukstanje);
          else
          {
            printf("nedefinisana promenljiva: ");
            stamime(iz->v.spromizr.promen);
            printf("\n");
            uradi();
          }
      staviarg(s);
      break;

    case lisizr: izraclistu(iz->v.slizr.argum, ar);
      break;

      case funizr:
        if (iz->v.sfunizr.oper > brugradjenih)
        {
          newar = vrhargstek;
          izraclistu(iz->v.sfunizr.argum, ar);
          primkorop(iz->v.sfunizr.oper,newar);
        }
        else
        {
          op = nalaziop(iz->v.sfunizr.oper);
          if (op>=ifop && op<=beginop)
              primkonop(op,iz->v.sfunizr.argum, ar);
          else
          {
            izraclistu(iz->v.sfunizr.argum, ar);
            primvrop(op);
          }
        }
      break;
  }
} /* izracunaj */
```

```
/* GLAVNI DEO - sekvencijane verzije */


/* vreme - meri vreme */
void vreme(void)
{
  vr2 = Time();
  vr2 = vr2 - vr1;
  miks = vr2 % 1000;
  vr2 = vr2 / 1000;
  ms = vr2 % 1000;
  vr2 = vr2 / 1000;
  sek = vr2 % 60;
  vr2 = vr2 / 60;
  m = vr2 % 60;
  s = vr2 / 60;
printf("\n %d sati %d minuta %d sekundi %d",
       " hiljaditih %d", "milionitih
\n",s,m,sek,ms,miks);
} /* vreme */

/* uradi - glavna petlja */
void uradi(void)
{
  while (!krajrada)
  {
    ucitaj();
    if (ististr(poz,4,"quit               ")) krajrada = 1;
     else if ((ulazpod[poz-1] = = '(') &&
       ((ististr(izbpraz(poz + 1),6,"define          ")) ||
       ( ististr(izbpraz(poz + 1),5,"defun           "))))
          {
            stamime(izdvfunkc());
            printf("\n");
          }
          else
          {
            vr1 = Time();
            tekizr = izdvizr();
            izracunaj(tekizr,0);
            stamsizr(uzmivrharg());
            izbaciarg(1);
            oslrefbr(tekizr);
            vreme();
            printf("\n");
            printf("\n");
          }
  }
  exit(0);
} /* uradi */


/*  PARALELNI DEO -   PRENOS ARGUMENATA
*/
```

```
listaizr primilistuizr(Channel *c); listaim
primilistuim(Channel *c);
listavr primilistuvr(Channel *c);
sizr primisizr(Channel *c);
void saljisizr(Channel *c, sizr y);
void saljilistuizr(Channel *c, listaizr y); void
saljiime(Channel *c, ime y);
void saljilistuim(Channel *c, listaim y); void
saljilistuvr(Channel *c, listavr y);

/* primisizr - prima s-izraz sa kanala c */
sizr primisizr(Channel *c)
{ sizr y;
  int s;

  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y = alocsizr(s);
    y->tsizr = s;
    y->refbr = 1;
    switch(s)
    {
      case nulasizr:
        y->v.snulasizr.gg = ChanInInt(c);
        break;

      case brsizr: y->v.sbrsizr.brvr = ChanInInt(c);
        break;

      case simsizr: y->v.ssimsizr.simvr =
primiime(c);
        break;

      case lsizr: y->v.slsizr.pocvr = primisizr(c);
        y->v.slsizr.ostvr = primisizr(c);
        povrefbr(y->v.slsizr.pocvr);
        povrefbr(y->v.slsizr.ostvr);
        break;
    }
  }
  return y;
} /* primisizr */

/* saljisizr - salje s-izraz na kanal c */
void saljisizr(Channel *c, sizr y)
{
  if (y = = NULL) ChanOutInt(c,NULP);
  else
  {
    ChanOutInt(c,y->tsizr);
    switch(y->tsizr)
    {
      case nulasizr: ChanOutInt(c, y->v.snulasizr.gg);
        break;

      case brsizr: ChanOutInt(c, y->v.sbrsizr.brvr);
        break;

      case simsizr: ChanOutInt(c, y-
>v.ssimsizr.simvr);
        break;

      case lsizr: saljisizr(c, y->v.slsizr.pocvr);
        saljisizr(c, y->v.slsizr.ostvr);
        break;
    }
  }
} /* saljisizr */

/* primiizraz - prima izraz sa kanala c */
izraz primiizraz(Channel *c)
{ izraz y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->itip = s;
    switch(s)
    {
      case brizr: y->v.sbrizr.siz = primisizr(c);
      break;

      case promizr:
        y->v.spromizr.promen = primiime(c);
        y->v.spromizr.ofset = ChanInInt(c);
        break;

      case funizr: y->v.sfunizr.oper = primiime(c);
        y->v.sfunizr.argum = primilistuizr(c);
        break;

      case lisizr: y->v.slizr.argum = primilistuizr(c);
        break;
    }
  }
  return y;
} /* primiizraz */

/* saljizraz - salje izraz na kanal c */
void saljiizraz(Channel *c, izraz y)
{
  if (y = = NULL) ChanOutInt(c,NULP);
  else
  {
    ChanOutInt(c,y->itip);
    switch(y->itip)
    {
      case brizr: saljisizr(c,y->v.sbrizr.siz);
        break;

      case promizr:
        saljiime(c, y->v.spromizr.promen);
        ChanOutInt(c, y->v.spromizr.ofset);
```

```
            break;

          case funizr: saljiime(c, y->v.sfunizr.oper);
            saljilistuizr(c, y->v.sfunizr.argum);
            break;

          case lisizr: saljilistuizr(c, y->v.slizr.argum);
            break;
        }
    }
} /* saljizraz */

/* primistanje - prima stanje sa kanala c */
stanje primistanje(Channel *c)
{ stanje y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->stprom = primilistuim(c);
    y->stvred = primilistuvr(c);
  }
  return y;
} /* primistanje */

/* saljistanje - salje stanje na kanal c */
void saljistanje(Channel *c, stanje y)
{
  if (y = = NULL) ChanOutInt(c,NULP);
  else
  {
    ChanOutInt(c,0);
    saljilistuim(c, y->stprom);
    saljilistuvr(c, y->stvred);
  }
} /* saljistanje */

/* primilistuizr - prima sa kanala c listu izraza */
listaizr primilistuizr(Channel *c)
{ listaizr y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = primiizraz(c);
    y->ostatak = primilistuizr(c);
  }
  return y;
} /* primilistuizr */

/* saljilistuizr - salje na kanal c listu izraza */
void saljilistuizr(Channel *c, listaizr y)
{
  if (y = = NULL)  ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljizraz(c, y->pocetak);
    saljilistuizr(c, y->ostatak);
  }
} /* saljilistuizr */

/* primiime - prima ime sa kanala c */
ime primiime(Channel *c)
{
  return ChanInInt(c);
} /* izracunaj */

/* saljiime - salje ime na kanal c */
void saljiime(Channel *c, ime y)
{
    ChanOutInt(c, y);
} /* izraclistu */

/* primilistuim - prima sa kanala c listu imena */
listaim primilistuim(Channel *c)
{ listaim y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = s;
    y->ostatak = primilistuim(c);
  }
  return y;
} /* primilistuim */

/* saljilistuim - salje listu imena na kanal c */
void saljilistuim(Channel *c, listaim y)
{
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    saljiime(c, y->pocetak);
    saljilistuim(c, y->ostatak);
  }
} /* saljilistuim */

/* primilistuvr - prima sa kanala c listu vrednosti */
listavr primilistuvr(Channel *c)
{ listavr y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = primisizr(c);
```

```
    y->ostatak = primilistuvr(c);
  }
  return y;
} /* primilistuvr */

/* saljilistuvr - salje na kanal c listu vrednosti */
void saljilistuvr(Channel *c, listavr y)
{
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljisizr(c, y->pocetak);
    saljilistuvr(c, y->ostatak);
  }
} /* saljilistuvr */

/* primideffun - prima definiciju funkcije sa kanala c
*/
deffun primideffun(Channel *c)
{ deffun y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->imefun = primiime(c);
    y->formarg = primilistuim(c);
    y->telofun = primiizraz(c);
    y->sleddeffun = primideffun(c);
  }
  return y;
} /* primideffun */

/* saljideffun - salje definiciju funkcije na kanal c */
void saljideffun(Channel *c, deffun y)
{
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljiime(c, y->imefun);
    saljilistuim(c, y->formarg);
    saljizraz(c, y->telofun);
    saljideffun(c, y->sleddeffun);
  }
} /* saljideffun */


/*  PARALELNI DEO -     GLAVNE FUNKCIJE */


void izrac(izraz iz, int ar);

/* odgovori - odgovara na poruke */
int odgovori()
{ Channel *c;
  int veza,i;
  int akcija;

  veza = 1;
  while (veza > -1 && veza < 4)
  {
    veza = ProcSkipAlt(LINK0IN, LINK1IN,
                       LINK2IN, LINK3IN, 0);
    if (veza > -1 && veza < 4)
    {
      c = ul[veza];
      akcija = ChanInInt(c);
      if (c != _boot_chan_in)
        if (akcija = = SLOBODNO)
          zauzet[veza] = 0; }
    }
} /* odgovori */

/* radi - izracunava vrednost izraza */
void radi(izraz iz, int ar, int *s, int *rac)
{ int s1,i;
  s1 = 0;

  odgovori();
  while((zauzet[s1] || pret[_node_number] = = s1)
        &&  s1 < BROJ_NASL+1)    s1++;
  if (s1<BROJ_NASL+1  &&
    node_number <=    IMA_NASL &&
    iz->itip==funizr &&
    iz->v.sfunizr.oper > brugradjenih)
    {
      zauzet[s1] = 1;
      ChanOutInt(izl[s1], RACUNAJ);
      odgovori();
      saljizraz(izl[s1], iz);
      ChanOutInt(izl[s1],ar);
      saljistanje(izl[s1], ukstanje);
      saljideffun(izl[s1], svedeffun);
      ChanOutInt(izl[s1], vrhargstek);
      for (i=0; i<vrhargstek; i++)
        saljisizr(izl[s1], argstek[i]);
      *rac = 1;
      *s = s1;
    }
    else
    {
      *rac = 0;
      izrac(iz,ar);
    }
}    /* radi */

/* uzmi - uzima vrednost od naslednika */
void uzmi(int br)
{ int sl;
  sizr gg;

  while (zauzet[br])  odgovori(); stani = 1;
  ChanOutInt(izl[br], ZAHTEV);
  sl = ProcAlt(ul[br], 0);
```

```
   ChanInInt(ul[br]);
   stani = 0;
   gg = primisizr(ul[br]);
   staviarg(gg);
} /* uzmi */

/* izraclistu - izracunava vrednost liste izraza */
void izraclistu(listaizr lizr, int ar)
{ int s,rac;

   if (lizr != NULL)
   {
      s=0;
      rac=0;
      radi(lizr->pocetak,ar,&s,&rac);
      izraclistu(lizr->ostatak, ar);
      if (rac) uzmi(s);
   }
} /* izraclistu */

/* izrac - izracunava vrednost izraza */
void izrac(izraz iz, int ar)
{
   izracunaj(iz, ar);
} /* izrac */

/* glavni - glavni deo interpretera */
void glavni()
{ int i;

   ulaz = stdin;
   postaviim();
   iniargstek();
   initmem();
   nulavr = alocsizr(nulasizr);
   povrefbr(nulavr);
   tacnavr = alocsizr(simsizr);
   tacnavr->v.ssimsizr.simvr = brimena;
   povrefbr(tacnavr);
   ukstanje = prazst();
   krajrada = 0;
   uradi();
   for(i=0; i<BROJ_NASL+1; i++)
      ChanOutInt(izl[i], KRAJRADA);
}

main()
{
_heapend = (void *) 0x803FFFFF;
   ProcToHigh();
   stani = 0;
   krajrada = 0;
   izlaz = fopen("trchapx1.izl","at");
   ul[0] = LINK0IN; izl[0] = LINK0OUT;
   ul[1] = LINK1IN; izl[1] = LINK1OUT;
   ul[2] = LINK2IN; izl[2] = LINK2OUT;
   ul[3] = LINK3IN; izl[3] = LINK3OUT;
   pret[1] = 0;  pret[2] = 1;  pret[3] = 2; pret[4] = 0;
   pret[5] = 2;  pret[6] = 0;  pret[7] = 1; pret[8] = 1;
   pret[9] = 2;  pret[10] = 0; pret[11] = 1; pret[12] =
1;
   pret[13] = 2; pret[14] = 0; pret[17] = 2; pret[18]
=0;
   pret[19] = 2;
   velst = -1;
   for (i=0; i<BROJ_NASL+1; i++) zauzet[i] = 0;
   glavni();
   fclose(ulaz);
   fclose(izlaz);
} /* main */
```

C.2. Kod koji se izvršava na ostalim transpjuterima

```
#include "stdio.h"
#include "stdlib.h"
#include "string.h"
#include "conc.h"


/* DEFINICIJE   */


#define BROJ_NASL 2
#define BROJ_TRANSP 17
#define IMA_NASL 8
#define RACUNAJ 1
#define ZAHTEV 2
#define VREDNOST 3
#define SLOBODNO 4
#define KRAJRADA 5
#define PORUKA 7
#define NULP -2
#define MAXST 10000
#define DUZPORUK 20
#define velmem 5000
#define duzimena 20     /* maksimalna duzina imena
*/ #define maximena 200 /* maksimalan broj
razlicitih imena */
#define maxulaz 2500 /* maksimalna duzina ulaza */
#define prompt "-> "
#define prompt2 "> "
#define komentar ';'
#define tabkod 9        /* ascii kod */
#define argstekvel 10000
typedef int  velimena;
typedef char imestr[duzimena+1];
typedef int number;
typedef int ime; /* a ime je indeks u nizimena */
typedef enum {ifop, condop, whileop, setop, setqop,
      beginop, plusop, minusop, timesop, divop,
      eqop, ltop, gtop, consop, carop, cdrop,
      numberpop, symbolpop, listpop, nullpop,
      printop } ugrop;
#define listaizr struct listaizrstr *
typedef int vrop;
typedef int konop;
#define sizr struct sizrstr *
typedef enum { nulasizr,brsizr,simsizr,lsizr } tipsizr;
#define listavr struct listavrstr *
typedef enum { brizr,promizr,funizr, lizr } tipizr;

typedef struct izrazstr
{
   tipizr itip;
   union
   {
      struct { sizr siz; } sbrizr;
      struct { ime promen; int ofset; } spromizr;
      struct {  ime oper; listaizr argum; } sfunizr;
      struct { listaizr argum; } slizr;
   } v;
} izrazstr;

#undef listavr
typedef struct listavrstr *listavr;
typedef struct sizrstr
{
   int refbr;
   tipsizr tsizr;
   union
   {
      struct { int gg; } snulasizr;
      struct { int brvr; } sbrsizr;
      struct { ime simvr; } ssimsizr;
      struct { sizr pocvr; sizr ostvr; } slsizr;
   } v;
} sizrstr;

typedef struct listavrstr
{
   sizr pocetak;
   listavr ostatak;
} listavrstr;

typedef izrazstr *izraz;
#define stanje struct stanjestr *
#define listaim struct listaimstr *
#define deffun struct deffunstr *
typedef struct listaizrstr
{
   izraz pocetak;
   listaizr ostatak;
} listaizrstr;

#undef deffun
typedef struct deffunstr *deffun;
typedef struct listaimstr
{
   ime pocetak;
   listaim ostatak;
} listaimstr;

typedef struct stanjestr
{
   listaim stprom;
   listavr stvred;
} stanjestr;

typedef struct deffunstr
{
   ime imefun;
```

```
  listaim formarg;
  izraz telofun;
  deffun sleddeffun;
} deffunstr;


/* DEKLARACIJA PROMENLJIVIH */


Channel *ul[4], *izl[4];
int krajrada, stani, rac;
int zauzet[BROJ_NASL+1];
sizr st[MAXST];
int velst,i;
int pret[20];
deffun svedeffun;
stanje ukstanje;
izraz tekizr;
sizr nulavr;
sizr tacnavr;
char ulazpod[maxulaz];
int duzulaza, poz;
imestr nizimena[ 200];
ime brimena, brugradjenih;
int quittingtime;
sizr argstek[argstekvel];
int vrhargstek;
sizr slobsizr;
int vr1,vr2,s,m,sek,ms,miks;


/* UPRAVLJANJE MEMORIJOM */


void uradi(void);

/* initmem - alocira memoriju za sve sizrstr
strukture */
void initmem(void)
{ int i;
  sizr s;

  slobsizr = NULL;
  for (i = 1; i <= velmem; i++)
  {
    s = slobsizr;
    slobsizr = malloc(sizeof(*slobsizr));
    slobsizr->v.slsizr.pocvr = s;
  }
} /* initmem */

/* alocsizr - alocira sizrstr strukturu iz liste slobodnih
sizraza */
sizr alocsizr(tipsizr t)
{ sizr s;

  s = slobsizr;
  s->tsizr = t;
  s->refbr = 0;
  slobsizr = slobsizr->v.slsizr.pocvr;
  return s;
} /* alocsizr */

/* vratisizr - vraca sizrstr u listu slobodnix sizraza  */
void vratisizr(sizr s)
{
  s->v.slsizr.pocvr = slobsizr;
  slobsizr = s;
} /* vratisizr */

/* povrefbr - povecava refbr polje argumenta */
void povrefbr(sizr s)
{
  s->refbr = s->refbr + 1;
} /* povrefbr */

/* smarefbr - smanjuje refbr polje argumenta */
void smarefbr(sizr s)
{
  s->refbr = s->refbr - 1;
  if (s->refbr = = 0)
  {
    if (s->tsizr = = lsizr)
    {
      smarefbr(s->v.slsizr.pocvr);
      smarefbr(s->v.slsizr.ostvr);
    }
    vratisizr(s);
  }
} /* smarefbr */

/* oslrefbr - smanjuje refbr u svim sizrazima izvan iz
*/
void oslrefbr(izraz iz)
{ listaizr argl;

  switch (iz->itip)
  {
    case brizr:   smarefbr(iz->v.sbrizr.siz);
      break;

    case promizr:  ;
      break;

    case funizr:
      argl = iz->v.sfunizr.argum;
      while (argl != NULL)
      {
        oslrefbr(argl->pocetak);
        argl = argl->ostatak;
      }
      break;
  }
} /* oslrefbr */

/* iniargstek - inicijalizuje argument stek     */
void iniargstek(void)
{
```

```
   vrhargstek = 1;
} /* iniargstek */

/* staviarg - stavlja jedan s-izraz na argument stek */
void staviarg(sizr s)
{
   argstek[vrhargstek-1] = s;
   vrhargstek = vrhargstek + 1;
   povrefbr(s);
} /* staviarg */

/* uzmivrharg - vraca vrh argument steka        */
sizr uzmivrharg(void)
{
   return argstek[vrhargstek - 2];
} /* uzmivrharg */

/* uzmidovrharg - vraca prvi s-izraz ispod vrha */
sizr uzmidovrharg(void)
{
if (vrhargstek > 2) return argstek[vrhargstek - 3];
} /* uzmidovrharg */

/* izbaciarg - izbacuje dati broj s-izaraza sa argument
steka */
void izbaciarg(int l)
{ int i;
   for (i = 1; i <= l; i++)
   {
      smarefbr(argstek[vrhargstek - 2]);
      vrhargstek = vrhargstek - 1;
   }
} /* izbaciarg */


/* PODRSKA OSNOVNIM TIPOVIMA
PODATAKA */


/* naplistaim - vraca listu imena sa pocetkom n i
ostatkom limena */
listaim naplistaim(ime im, listaim limena)
{  listaim novalimena;

   novalimena = malloc(sizeof(*novalimena));
   novalimena->pocetak = im;
   novalimena->ostatak = limena;
   return novalimena;
}  /* naplistaim */

/* naplistavr - vraca listu vrednosti sa pocetkom n i
ostatkom lvred */
listavr naplistavr(sizr n, listavr lvred)
{  listavr novalvred;

      novalvred = malloc(sizeof(*novalvred));
      novalvred->pocetak = n;
      novalvred->ostatak = lvred;
      return novalvred;
}  /* naplistavr */

/* napstanje - vraca stanje sa promenljivima limena i
vrednostima lvred */
stanje napstanje(listaim limena, listavr lvred)
{  stanje rho;

   rho = malloc(sizeof(*rho));
   rho->stprom = limena;
   rho->stvred = lvred;
   return rho;
}  /* napstanje */

/* duzlistavr - vraca duzinu liste vrednosti lvred */
int duzlistavr(listavr lvred)
{  int i;

   i = 0;
   while (lvred != NULL)
   {
      i = i + 1;
      lvred = lvred->ostatak;
   }
   return i;
}  /* duzlistavr */

/* duzlistaim - vraca duzinu liste imena limena */
int duzlistaim(listaim limena)
{  int i;

   i = 0;
   while (limena != NULL)
   {
      i = i + 1;
      limena = limena->ostatak;
   }
   return i;
}  /* duzlistaim */


/* OPERACIJE NAD IMENIMA FUNKCIJA       */


/* uzmideffun - uzima definiciju funkcije fime iz
svedeffun */
deffun uzmideffun(ime fime)
{  deffun f;
   int nasao;

   nasao = 0;
   f = svedeffun;
   while ((f != NULL) && !nasao)
      if (f->imefun = = fime) nasao = 1;
      else f = f->sleddeffun;
   return f;
}  /* uzmideffun */

/* postaviim - postavlja sva definisana imena u
nizimena */
```

```
void postaviim(void)
{ int i;

  strcpy(ulazpod,"              ");
  svedeffun = NULL;
  i = 1;
  strcpy(nizimena[i-1],"if               "); i = i + 1;
  strcpy(nizimena[i-1],"cond             "); i = i + 1;
  strcpy(nizimena[i-1],"while            "); i = i + 1;
  strcpy(nizimena[i-1],"set              "); i = i + 1;
  strcpy(nizimena[i-1],"setq             "); i = i + 1;
  strcpy(nizimena[i-1],"begin          "); i = i + 1;
  strcpy(nizimena[i-1],"+                "); i = i + 1;
  strcpy(nizimena[i-1],"-              "); i = i + 1;
  strcpy(nizimena[i-1],"*              "); i = i + 1;
  strcpy(nizimena[i-1],"/                "); i = i + 1;
  strcpy(nizimena[i-1],"=                "); i = i + 1;
  strcpy(nizimena[i-1],"<                "); i = i + 1;
  strcpy(nizimena[i-1],">                "); i = i + 1;
  strcpy(nizimena[i-1],"cons             "); i = i + 1;
  strcpy(nizimena[i-1],"car            "); i = i + 1;
  strcpy(nizimena[i-1],"cdr            "); i = i + 1;
  strcpy(nizimena[i-1],"number?          "); i = i + 1;
  strcpy(nizimena[i-1],"symbol?          "); i = i + 1;
  strcpy(nizimena[i-1],"list?            "); i = i + 1;
  strcpy(nizimena[i-1],"null?          "); i = i + 1;
  strcpy(nizimena[i-1],"print            "); i = i + 1;
  strcpy(nizimena[i-1],"T                "); 
  brimena = i;
  brugradjenih = i;
} /* postaviim */

/* nalaziop - nalazi za ime odgovarajuci korop */
ugrop nalaziop(ime oper)
{ ugrop op;
  int i;

  op = ifop;
  for (i = 1; i <= oper - 1; i++) op++;
  return op;
} /* nalaziop */


/* POSTAVLJANJE PARAMETARA */


/* prazst - vraca prazno stanje */
stanje prazst(void)
{
    return napstanje(NULL,NULL);
} /* prazst */

/* postprom - postavlja promenljivoj im vrednost n u
stanju rho */
void postprom(ime im, sizr s, stanje rho)
{
  povrefbr(s);
  rho->stprom = naplistaim(im,rho->stprom);
  rho->stvred = naplistavr(s,rho->stvred);
} /* postprom */

/* nadjiprom - nalazi promenljivu im u stanju rho */
listavr nadjiprom(ime im,      stanje rho)
{ listaim limena;
  listavr lvred;
  int nasao;

  nasao = 0;
  limena = rho->stprom;
  lvred = rho->stvred;
  while ((limena != NULL) && !nasao)
    if (limena->pocetak = = im) nasao = 1;
    else
    {
      limena = limena->ostatak;
      lvred = lvred->ostatak;
    }
    return lvred;
} /* nadjiprom */

/* dodprom - dodeli promenljivoj im vrednost n u
stanju rho */
void dodprom(ime im, sizr s, stanje rho)
{ listavr varloc;

  povrefbr(s);
  varloc = nadjiprom(im,rho);
  smarefbr(varloc->pocetak);
  varloc->pocetak = s;
} /* dodprom */

/* vrprom - vraca vrednost promenljive im u stanju
rho */
sizr vrprom(ime im, stanje rho)
{ listavr lvred;

  lvred = nadjiprom(im,rho);
  return lvred->pocetak;
} /* vrprom */

/* dalije_prom - da li je dato ime promenljiva ili ne u
stanju rho */
int dalije_prom(ime im, stanje rho)
{
  return nadjiprom(im,rho) != NULL;
} /* dalije_prom */


/* MANIPULACIJA S-IZRAZIMA */


/* dalije_tacan - vraca 1 ako nije nulti s-izraz */
int dalije_tacan(sizr s)
{
  return s->tsizr != nulasizr;
} /* dalije_tacan */

/* primvrop - primenjuje datu operaciju */
```

```
void primoper(int n1, int n2, vrop op, sizr *rezultat)
{ sizr pom;

  pom = alocsizr(brsizr);
  switch (op)
  {
    case plusop:  pom->v.sbrsizr.brvr = n1 + n2;
      break;

    case minusop:  pom->v.sbrsizr.brvr = n1 - n2;
      break;

    case timesop:  pom->v.sbrsizr.brvr = n1 * n2;
      break;

    case divop:   pom->v.sbrsizr.brvr = n1 / n2;
      break;
  }
  *rezultat = pom;
} /* primoper */

/* primrel - primeni relaciju na argumente */
void primrel(int n1, int n2, vrop op, sizr *rezultat) {
  switch (op)
  {
    case ltop:   if (n1 < n2) *rezultat = tacnavr;
      break;

    case gtop:   if (n1 > n2) *rezultat = tacnavr;
      break;
  }
} /* primrel */

/* brargum - nalazi broj argumenata operacije */
int brargum(vrop op)
{
  if (op>=plusop && op<=consop) return 2;
  else return 1;
} /* brargum */

/* primvrop - primenjuje datu operaciju */
void primvrop(vrop op)
{ sizr rezultat;
  sizr s1;
  sizr s2;

  rezultat = nulavr;
  s1 = uzmivrharg();
  if (brargum(op) = = 2)
  {
    s2 = s1;
    s1 = uzmidovrharg();
  }
  if ((op>=plusop && op<=divop) ||
  (op>=ltop && op<=gtop))
    if ((s1->tsizr = = brsizr) &&
      (s2->tsizr = = brsizr))
      if (op>=plusop && op<=divop)
        primoper(s1->v.sbrsizr.brvr,
                 s2->v.sbrsizr.brvr,    op,&rezultat);
      else primrel(s1->v.sbrsizr.brvr,
                 s2->v.sbrsizr.brvr,    op, &rezultat);
    else
      switch (op)
      {
        case eqop:
          if ((s1->tsizr = = nulasizr) &&
          (s2->tsizr = = nulasizr))
            rezultat = tacnavr;
          else if ((s1->tsizr = = brsizr) &&
              (s2->tsizr = = brsizr) &&
              (s1->v.sbrsizr.brvr = =
              s2->v.sbrsizr.brvr))
                rezultat = tacnavr;
              else if ((s1->tsizr = = simsizr) &&
                  (s2->tsizr = = simsizr) &&
                  (s1->v.ssimsizr.simvr = =
                  s2-> v.ssimsizr.simvr))
                    rezultat = tacnavr;
          break;

        case consop:
          rezultat = alocsizr(lsizr);
          rezultat->v.slsizr.pocvr = s1;
          povrefbr(s1);
          rezultat->v.slsizr.ostvr = s2;
          povrefbr(s2);
          break;

        case carop:
          rezultat = s1->v.slsizr.pocvr;
          break;

        case cdrop:
          rezultat = s1->v.slsizr.ostvr;
          break;

        case numberpop:
          if (s1->tsizr = = brsizr) rezultat = tacnavr;
          break;

        case symbolpop:
          if (s1->tsizr = = simsizr) rezultat =
tacnavr;
          break;

        case listpop:
          if (s1->tsizr = = lsizr) rezultat = tacnavr;
          break;

        case nullpop:
          if (s1->tsizr = = nulasizr)
            rezultat = tacnavr;
          break;
      }
  izbaciarg(brargum(op));
  staviarg(rezultat);
} /* primvrop */
```

```
/* IZRACUNAVANJE VREDNOSTI IZRAZA */


void izracunaj(izraz iz, int ar);
void izraclistu(listaizr lizr, int ar);

/* primkorop - primenjuje korisnicki definisanu
funkciju */
void primkorop(ime im, int newar)
{  deffun f;
   sizr s;

   f = uzmideffun(im);
   izracunaj(f->telofun,newar);
   s = uzmivrharg();
   vrhargstek = vrhargstek - 1;
   izbaciarg(duzlistaim(f->formarg));
   argstek[vrhargstek-1] = s;
   vrhargstek = vrhargstek + 1;
} /* primkorop */

/* primkonop - primenjuje kontrolnu operaciju */
void primkonop(konop op, listaizr argum, int ar)
{ sizr s;

   switch (op)
   {
      case ifop:
         izracunaj(argum->pocetak,ar);
         s = uzmivrharg();
         if (dalije_tacan(s))
         {
            izbaciarg(1);
            izracunaj(argum->ostatak->pocetak,ar);
         }
         else
         {
            izbaciarg(1);
            izracunaj(argum->ostatak->ostatak
                  ->pocetak,ar);
         }
         break;

      case condop:
         izracunaj(argum->pocetak->v.slizr.argum
                  ->pocetak, ar);
         s = uzmivrharg();
         while((argum->ostatak != NULL) &&
         !dalije_tacan(s))
         {
            izracunaj(argum->pocetak
                     ->v.slizr.argum->pocetak, ar);
            s = uzmivrharg();
            if (!dalije_tacan(s))
               argum = argum->ostatak;
         }
         if (argum->ostatak != NULL)
         {
            izracunaj(argum->pocetak->v.slizr.argum
                     -> ostatak->pocetak, ar);
            s = uzmivrharg();
         }
         else
         {
            izracunaj(argum->pocetak->
                  v.slizr.argum->pocetak, ar);
            s = uzmivrharg();
            if (dalije_tacan(s))
            {
               izracunaj(argum->pocetak->v.slizr.argum
                        -> ostatak->pocetak, ar);
               s = uzmivrharg();
            }
            staviarg(s);
         }
         break;

      case whileop:
         staviarg(nulavr);
         izracunaj(argum->pocetak,ar);
         s = uzmivrharg();
         while (dalije_tacan(s))
         {
            izbaciarg(2);
            izracunaj(argum->ostatak->pocetak,ar);
             izracunaj(argum->pocetak,ar);
            s = uzmivrharg();
         }
         izbaciarg(1);
         break;

      case setop:
         izracunaj(argum->ostatak->pocetak,ar);
         s = uzmivrharg();
         if (argum->pocetak->v.spromizr.ofset > 0)
         {
            povrefbr(s);
            smarefbr(argstek[ar + argum->pocetak->
                     v.spromizr.ofset - 2]);
            argstek[ar + argum->pocetak->
                  v.spromizr.ofset - 2] = s;
         }
         else if (dalije_prom(argum->pocetak->
               v.spromizr.promen,ukstanje))
               dodprom(argum->pocetak->
                     v.spromizr.promen, s,ukstanje);
               else postprom(argum->pocetak->
                     v.spromizr.promen,s,ukstanje);
         break;

      case setqop:
         izracunaj(argum->ostatak->pocetak,ar);
         s = uzmivrharg();
         if (argum->pocetak->v.spromizr.ofset > 0)
         {
            povrefbr(s);
```

```
        smarefbr(argstek[ar + argum->pocetak->
                v.spromizr.ofset - 2]);
        argstek[ar + argum->pocetak->
                v.spromizr.ofset - 2] = s;
      }
      else if (dalije_prom(argum->pocetak->
          v.spromizr.promen,ukstanje))
          dodprom(argum->pocetak->
                v.spromizr.promen, s,ukstanje);
          else postprom(argum->pocetak->
                    v.spromizr.promen,
s,ukstanje);
      break;

    case beginop:
      while (argum->ostatak != NULL)
      {
        izracunaj(argum->pocetak,ar);
        izbaciarg(1);
        argum = argum->ostatak;
      }
      izracunaj(argum->pocetak,ar);
      break;
  }
} /* primkonop */

/* izracunaj - izracunava vrednost izraza */
void izracunaj(izraz iz, int ar)
{ sizr s;
  ugrop op;
  int newar;

  switch (iz->itip)
  {
    case brizr: staviarg(iz->v.sbrizr.siz);
      break;

    case promizr:
      if (iz->v.spromizr.ofset > 0)
        s = argstek[ar + iz->v.spromizr.ofset - 2];
      else if (dalije_prom(iz->v.spromizr.promen,
          ukstanje))
            s = vrprom(iz->v.spromizr.promen,
                    ukstanje);
      staviarg(s);
      break;

    case lizr: izraclistu(iz->v.slizr.argum, ar);
      break;

    case funizr:
      if (iz->v.sfunizr.oper > brugradjenih)
      {
        newar = vrhargstek;
        izraclistu(iz->v.sfunizr.argum, ar);
        primkorop(iz->v.sfunizr.oper,newar);
      }
      else
      {
        op = nalaziop(iz->v.sfunizr.oper);
        if (op>=ifop && op<=beginop)
          primkonop(op,iz->v.sfunizr.argum, ar);
        else
        {
          izraclistu(iz->v.sfunizr.argum, ar);
          primvrop(op);
        }
      }
      break;
  }
} /* izracunaj */


/* PARALELNI DEO - PRENOS ARGUMENATA
*/


listaizr primilistuizr(Channel *c);
listaim primilistuim(Channel *c);
listavr primilistvr(Channel *c);
sizr primisizr(Channel *c);
void saljisizr(Channel *c, sizr y);
void saljilistuizr(Channel *c, listaizr y); void
saljiime(Channel *c, ime y);
void saljilistuim(Channel *c, listaim y); void
saljilistvr(Channel *c, listavr y); ime
primiime(Channel *c);
void saljiime(Channel *c, ime y);

/* primisizr - prima s-izraz sa kanala c */
sizr primisizr(Channel *c)
{ sizr y;
  int s;

  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y = alocsizr(s);
    y->tsizr = s;
    y->refbr = 1;
    switch(s)
    {
      case nulasizr:
        y->v.snulasizr.gg = ChanInInt(c);
        break;

      case brsizr: y->v.sbrsizr.brvr = ChanInInt(c);
        break;

      case simsizr:
        y->v.ssimsizr.simvr = primiime(c);
        break;

      case lsizr: y->v.slsizr.pocvr = primisizr(c);
        povrefbr(y->v.slsizr.pocvr);
        y->v.slsizr.ostvr = primisizr(c);
        povrefbr(y->v.slsizr.ostvr);
```

```
            break;
        }
    }
    return y;
}    /* primisizr */

/* saljisizr - salje s-izraz na kanal c */
void saljisizr(Channel *c, sizr y)
{
    if (y = = NULL) ChanOutInt(c,NULP);
    else
    {
        ChanOutInt(c,y->tsizr);
        switch(y->tsizr)
        {
            case nulasizr:
                ChanOutInt(c, y->v.snulasizr.gg);
                break;

            case brsizr: ChanOutInt(c, y->v.sbrsizr.brvr);
                break;

            case simsizr:
                ChanOutInt(c, y->v.ssimsizr.simvr);
                break;

            case lsizr: saljisizr(c, y->v.slsizr.pocvr);
                saljisizr(c, y->v.slsizr.ostvr);
                break;
        }
    }
} /* saljisizr */

/* primiizraz - prima izraz sa kanala c */
izraz primiizraz(Channel *c)
{  izraz y;
    int s;

    y = malloc(sizeof(*y));
    s = ChanInInt(c);
    if (s = = NULP) y = NULL;
    else
    {
        y->itip = s;
        switch(s)
        {
            case brizr: y->v.sbrizr.siz = primisizr(c);
                break;

            case promizr:
                y->v.spromizr.promen = primiime(c);
                y->v.spromizr.ofset = ChanInInt(c);
                break;

            case funizr: y->v.sfunizr.oper = primiime(c);
                y->v.sfunizr.argum = primilistuizr(c);
                break;

            case lizr: y->v.slizr.argum = primilistuizr(c);
                break;
        }
    }
    return y;
}    /* primiizraz */

/* saljiizraz - salje izraz na kanal c */
void saljiizraz(Channel *c, izraz y)
{
    if (y = = NULL) ChanOutInt(c,NULP);
    else
    {
        ChanOutInt(c,y->itip);
        switch(y->itip)
        {
            case brizr: saljisizr(c,y->v.sbrizr.siz);
                break;

            case promizr:
                saljiime(c, y->v.spromizr.promen);
                ChanOutInt(c, y->v.spromizr.ofset);
                break;

            case funizr:   saljiime(c, y->v.sfunizr.oper);
                saljilistuizr(c, y->v.sfunizr.argum); break;

            case lizr: saljilistuizr(c, y->v.slizr.argum);
                break;
        }
    }
} /* saljiizraz */

/* primistanje - prima stanje sa kanala c */
stanje primistanje(Channel *c)
{  stanje y;
    int s;

    y = malloc(sizeof(*y));
    s = ChanInInt(c);
    if (s = = NULP) y = NULL;
    else
    {
        y->stprom = primilistuim(c);
        y->stvred = primilistvr(c);
    }
    return y;
} /* primistanje */

/* saljistanje - salje stanje na kanal c */
void saljistanje(Channel *c, stanje y)
{
    if (y = = NULL) ChanOutInt(c,NULP);
    else
    {
        ChanOutInt(c,0);
        saljilistuim(c, y->stprom);
        saljilistvr(c, y->stvred);
    }
} /* saljistanje */
```

```
/* primilistuizr - prima sa kanala c listu izraza */
listaizr primilistuizr(Channel *c)
{ listaizr y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = primiizraz(c);
    y->ostatak = primilistuizr(c);
  }
  return y;
} /* primilistuizr */

/* saljilistuizr - salje na kanal c listu izraza */
void saljilistuizr(Channel *c, listaizr y)
{
  if (y = = NULL)  ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljiizraz(c, y->pocetak);
    saljilistuizr(c, y->ostatak);
  }
} /* saljilistuizr */

/* primiime - prima ime sa kanala c */
ime primiime(Channel *c)
{
    return ChanInInt(c);
} /* primiime */

/* saljiime - salje ime na kanal c */
void saljiime(Channel *c, ime y)
{
  ChanOutInt(c, y);
} /* saljiime */

/* primilistuim - prima sa kanala c listu imena */
listaim primilistuim(Channel *c)
{ listaim y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = s;
    y->ostatak = primilistuim(c);
  }
  return y;
} /* primilistuim */

/* saljilistuim - salje listu imena na kanal c */
void saljilistuim(Channel *c, listaim y)
{
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    saljiime(c, y->pocetak);
    saljilistuim(c, y->ostatak);
  }
} /* saljilistuim */

/* primilistuvr - prima sa kanala c listu vrednosti */
listavr primilistuvr(Channel *c)
{ listavr y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->pocetak = primisizr(c);
    y->ostatak = primilistvr(c);
  }
  return y;
} /* primilistvr */

/* saljilistuvr - salje na kanal c listu vrednosti */
void saljilistuvr(Channel *c, listavr y)
{
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljisizr(c, y->pocetak);
    saljilistvr(c, y->ostatak);
  }
} /* saljilistvr */

/* primideffun - prima definiciju funkcije sa kanala c
*/
deffun primideffun(Channel *c)
{ deffun y;
  int s;

  y = malloc(sizeof(*y));
  s = ChanInInt(c);
  if (s = = NULP) y = NULL;
  else
  {
    y->imefun = primiime(c);
    y->formarg = primilistuim(c);
    y->telofun = primiizraz(c);
    y->sleddeffun = primideffun(c);
  }
  return y;
} /* primideffun */

/* saljideffun - salje definiciju funkcije na kanal c */
void saljideffun(Channel *c, deffun y)
{
```

```
  if (y = = NULL) ChanOutInt(c, NULP);
  else
  {
    ChanOutInt(c, 0);
    saljiime(c, y->imefun);
    saljilistuim(c, y->formarg);
    saljiizraz(c, y->telofun);
    saljideffun(c, y->sleddeffun);
  }
}    /* saljideffun */

/* uzmisizr - uzmi s-izraz sa steka za komunikaciju */
sizr uzmisizr()
{ sizr gg;

  gg = st[velst];
  velst--;
  return gg;
} /*  uzmisizr  */

/* stavisizr - stavlja s-izraz na stek za komunikaciju
*/
void stavisizr(sizr x)
{
  velst++;
  st[velst] = x;
}    /*  stavisizr  */


/* PARALELNI DEO      -  GLAVNE FUNKCIJE */


void izrac(izraz iz, int ar);

/* odgporuke - odgovara na poruke naslednika */
void odgporuke()
{ Channel *c;
  int veza,i, gg;
  int akcija;

  veza = 1;
  while (veza > -1    && veza<4)
  {
    veza = ProcSkipAlt(LINK0IN, LINK1IN,
                       LINK2IN, LINK3IN, 0);
    if (veza > -1  && veza < 4)
    {
      c = ul[veza];
      akcija = ChanInInt(c);
      if (c = = _boot_chan_in)
      {
        switch(akcija)
        {
          case ZAHTEV:
            ChanOutInt(_boot_chan_out,
                       VREDNOST);
            saljisizr(_boot_chan_out, uzmisizr());
            break;
          case KRAJRADA:
            krajrada = 1;
            for(i = 0; i < BROJ_NASL+1; i++)
              if (pret[_node_number] != i)
                ChanOutInt(izl[i], KRAJRADA);
            break;
        }
      }
      else
        if (akcija = = SLOBODNO)
          zauzet[veza] = 0;
    }
  }
} /* odgporuke */

/* odgovori - odgovara na poruke */

void odgovori()
{ Channel *c;
  int veza,i,gg;
  izraz po1;
  int akcija;
  int ar;

  krajrada = 0;
  veza = 1;
  while (!krajrada)
  {
    veza = ProcSkipAlt(LINK0IN, LINK1IN,
                       LINK2IN, LINK3IN, 0);
    if (veza > -1  && veza < 4)
    {
      c = ul[veza];
      akcija = ChanInInt(c);
      if (c = = _boot_chan_in)
      {
        switch(akcija)
        {
          case RACUNAJ:
            po1 = primiizraz(c);
            ar = ChanInInt(c);
            ukstanje = primistanje(c);
            svedeffun = primideffun(c);
            vrhargstek = ChanInInt(c);
            for (i=0; i<vrhargstek; i++)
              argstek[i] = primisizr(c);
            izrac(po1, ar);
            break;

          case ZAHTEV:
            ChanOutInt(_boot_chan_out,
                       VREDNOST);
            saljisizr(_boot_chan_out, uzmisizr());
            break;

          case KRAJRADA:
            krajrada = 1;
            for(i = 0; i < BROJ_NASL+1; i++)
              if (pret[_node_number] != i)
```

```
                    ChanOutInt(izl[i], KRAJRADA);
                break;
            }
          }
          else
            if (akcija = = SLOBODNO)
              zauzet[veza] = 0;
      }
    }
} /* odgovori */

/* radi - izracunava vrednost izraza */
void radi(izraz iz, int ar, int *s, int *rac)
{ int s1,i;

  s1 = 0;
  while((zauzet[s1] || pret[_node_number] = = s1)
        && s1 < BROJ_NASL+1) s1++;
  if (s1<BROJ_NASL+1  &&
    _node_number<=IMA_NASL &&
    iz->itip==funizr  &&
    iz->v.sfunizr.oper > brugradjenih)
  {
    zauzet[s1] = 1;
    ChanOutInt(izl[s1], RACUNAJ);
    saljiizraz(izl[s1], iz);
    ChanOutInt(izl[s1], ar);
    saljistanje(izl[s1], ukstanje);
    saljideffun(izl[s1], svedeffun);
    ChanOutInt(izl[s1], vrhargstek);
    for (i=0; i<vrhargstek; i++)
      saljisizr(izl[s1], argstek[i]);
    *rac = 1;
    *s = s1;
  }
  else
  {
    *rac = 0;
    *s = s1;
    izracunaj(iz,ar);
  }
} /* radi */

/* uzmi - uzima vrednost sa steka naslednika */
void uzmi(int br)
{ int sl;
  sizr gg;

  while (zauzet[br]) odgporuke();
  stani = 1;
  ChanOutInt(izl[br], ZAHTEV);
  sl = ProcAlt(ul[br], 0);
  ChanInInt(ul[br]);
  gg = primisizr(ul[br]);
  stani = 0;
  staviarg(gg);
} /* uzmi */

/* izraclistu - izracunava vrednost liste izraza */
void izraclistu(listaizr lizr, int ar)
{ int s, rac;

  if (lizr = = NULL) return NULL;
  else
  {
    rac=0;
    s=0;
    radi(lizr->pocetak, ar, &s, &rac);
    izraclistu(lizr->ostatak, ar);
    if (rac) uzmi(s);
  }
}   /* izraclistu */

/* izrac - izracunava vrednost izraza */
void izrac(izraz iz, int ar)
{ sizr h;

  izracunaj(iz, ar);
  h = uzmivrharg();
  stavisizr(h);
  ChanOutInt(_boot_chan_out, SLOBODNO);
}   /* izrac */

/* glavni - glavni deo */
void glavni()
{
  postaviim();
  iniargstek();
  initmem();
  nulavr = alocsizr(nulasizr);
  povrefbr(nulavr);
  tacnavr = alocsizr(simsizr);
  tacnavr->v.ssimsizr.simvr = brimena;
  povrefbr(tacnavr);
} /* glavni */

main()
{
  _heapend = (void *) 0x803FFFFF;
  ProcToHigh();
  stani = 0;
  krajrada = 0;
  ul[0] = LINK0IN; izl[0] = LINK0OUT;
  ul[1] = LINK1IN; izl[1] = LINK1OUT;
  ul[2] = LINK2IN; izl[2] = LINK2OUT;
  ul[3] = LINK3IN; izl[3] = LINK3OUT;
  pret[1] = 0;  pret[2] = 1;  pret[3] = 2; pret[4] = 0;
  pret[5] = 2;  pret[6] = 0;  pret[7] = 1; pret[8] = 1;
  pret[9] = 2;  pret[10] = 0; pret[11] = 1; pret[12] =
1;
  pret[13] = 2; pret[14] = 0; pret[17] = 2; pret[18]
=0;
  pret[19] = 2;
  velst = -1;
  for(i=0; i<BROJ_NASL+1; i++) zauzet[i] = 0;
  glavni();
  odgovori();
}
```

Literatura

[1] **Kamin N. S.**, "*Programming languages - An interpreter based approach"*, Addison-Wesley, 1990.

[2] **Akl S.**, "*Design and analysis of paralel algorihms"*, Prentice-Hall International, 1989.

[3] **Dershem H.L., Jipping M.J.** *"Functional model", Programming languages, structures and models*, p.p. 289-322, Wadsworth Publishing Company, 1990.

[4] **Budimac Z., Ivanović M., Tošić D., Putnik Z.**, "*LISP kroz primere"*, Univerzitet u Novom Sadu - Institut za matematiku, 1991.

[5] **Brassard G., Bratley B.,** *"Algorithmics, Theory and Practice"*, Prentice-Hall International, 1988.

[6] **Mock J.**, "*Processes, Channels, and Semaphores", Trasputer Toolset*, Inmos Corp., 1989.

[7] "*PP C Preprocesor User Guide", Transputer Toolset*, Inmos Corp., 1989.

[8] "*TCX Transputer C Compiler User Guide", Transputer Toolset*, Inmos Corp. 1989.

[9] *"TASM Transputer Assembler User Guide", Transputer Toolset*, Inmos Corp. 1989.

[10] *"TLNK Transputer Linker User Guide", Transputer Toolset*, Inmos Corp. 1989.

[11] "*LD-ONE Loader User Guide", Transputer Toolset*, Inmos Corp. 1989.

[12] *"LD-NET Network Loader User Guide", Transputer Toolset*, Inmos Corp. 1989.

[13] "*TLIB Transputer Librarian User Guide", Transputer Toolset,* Inmos Corp. 1989.

[14] "*Transputer C Library Description",* Inmos Corp. 1989.

[15] **Flynn M.J.** "*Very High-Speed Computing Systems", Proc IEEE,* vol. 54, 1966., pp. 1901.-9.

[16] **Cvetković D., Simić S.** *"Kombinatorika - Klasična i moderna"*, Naučna knjiga, 1990.

[17] **Ortega J.M.** *"Introduction to parallel and vector solution of linear systems",* Plenum Pul Corporation, 1988.

[18] **Babb R. G.** "*Programming Parallel Processors",* Addison-Wesley, 1988.

[19] **Ben-Ari M.** *"Principles of Concurent and Distributed Programming"*, Prentice Hall International, 1988.

[20] **Cormen T.H., Leiserson C.E., Rivest R.L.** "*Introduction to Algorithms*", MIT Press - McGraw Hill, 1991.

[21] **Ashcroft E.A., Faustini A.A., Jagannathan R.** "*An Intensional Language for Parallel Applications Programming*", *Parallel Functional Languages and Compilers,* p.p. 11.-50. , ACM Press, 1991.

[22] **Hudak P.** "*Para-Functional Programming in Haskell*", *Parallel Functional Languages and Compilers,* p.p. 159.-196. , ACM Press, 1991.

[23] **Mou Z.G.** "*A Formal Model for Divide-and-Conquer and Its Paralell Realization",* PhD thesis, Yale University, Department of Computer Science, 1990.

[24] **Chen M., Young-il Choo, Jingke Li** "*Crystal: Theory and Pragmatics of Generating Efficient Parallel Code", Parallel Functional Languages and Compilers,* p.p. 255.-308. , ACM Press, 1991.

[25] **Sarkar V.** "*PTRAN - The IBM Parallel Translation System" Parallel Functional Languages and Compilers,* p.p. 309.-392. , ACM Press, 1991.

[26] **Skedzielewski S.K., Glauert J.** "*IF1- an intermediate form for aplicative languages"*, Manual M-170, Lawrence Livermore National Laboratory, 1985.

[27] **Szymanski K. B.** "*Conclusion", Parallel Functional Languages and Compilers,* p.p. 393.-409., ACM Press, 1991.

[28] **Skedzielewski S.K.** "*Sisal", Parallel Functional Languages and Compilers,* p.p. 105.-158., ACM Press, 1991.

[29] **Pushpa R.** "*An Equational Language for Data-Parallelism", Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming,* p.p. 112.-118., ACM Press, 1993.

[30] **Leung S., Zahorjan J.** *"Improving the Performance of Runtime Paralelization", Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming,* p.p. 83.-91., ACM Press, 1993.

[31] **Wagner D.B., Calder D.B.** *"Leapfrogging: A Portable Technique for Implementating Efficient Futures", Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming,* p.p. 208.-217., ACM Press, 1993.

[32] **Burn G.L., Hankin C., Abramsky S.** *"Strictness analysis for higher order function", Sci.-Comput.-Programming 7.,no. 3,* p.p. 249.-278., 1986.

[33] **Kontothansis L., Wisniewski R.W.** "*Using Scheduler Information to Achieve Optimal Barrier Synchronization Performance", Proceedings of the Fourth ACM SIGPLAN Symposium on Principles & Practice of Parallel Programming,* p.p. 64.-72., ACM Press, 1993.