Nikola Milosavljević

Algoritam za uređivanje procedura i funkcija Paskal programa u optimalan redosled

Prilikom pisanja većih Pascal programa, ili slaganja programa iz više nezavisnih tekst fajlova, javlja se problem uređivanja procedura i funkcija u takav redosled koji omogućava ispravno prevođenje programa. U radu je dat opis programa koji izvodi automatsko raspoređivanje procedura i funkcija, tako da rezultujući program ima njihov ispravan redosled, ili dodate `forward` *deklaracije na onim mestima gde je to neophodno.*

Uvod

Pod optimalnim redosledom se podrazumeva onaj redosled procedura i funkcija kod koga svaka procedura poziva samo one procedure koje su prethodno deklarisane. U slučaju uzajamne rekurzije nije moguće jednostavno presložiti procedure. Tada će program deklarisati minimalan broj procedura kao `forward`, tako da program na kraju ipak proradi onako kako je programer želeo. Kada je procedura deklarisana kao `forward`, njen kod se ne nalazi odmah iza deklaracije, već negde dalje u programu. `forward` deklaracija ukazuje kompajleru da u nastavku programa može očekivati stvarni kod te procedure. Algoritam obuhvata i procedure deklarisane unutar drugih procedura, jer je u osnovi rekurzivan.

Tokenizacija, traženje deklaracija i poziva procedura u kodu

U ovoj fazi ulazna datoteka se čita znak po znak i na osnovu definisanog skupa znakova koji predstavljaju delimitere, izdvajaju se tokeni. Oni se smeštaju u jednostruko povezanu listu. Registruju se i svi CR/LF znakovi, a početak i kraj svakog reda se zapisuju u listu. Ti podaci se kasnije koriste kod translacije brojeva linija. Podaci o procedurama se čuvaju u

Nikola Milosavljević (1978), Požega, Radosava kovačevića 3, učenik 3. razreda Gimanzije „Sveti Sava" u Požegi

strukturi oblika stabla, čiji su čvorovi procedure (koren predstavlja glavni program). Program prolazi kroz listu tokena, i kada naiđe na rezervisanu reč `procedure` ili `function`, dinamički alocira memoriju za novi list u stablu. Kada se detektuje deklaracija procedure, pretražuje se njen kod kako bi se otkrile njene potprocedure (sledeći nivo rekurzije). U svakom čvoru se pamti mesto početka deklaracije, mesto početnog `begin`-a i krajnjeg `end`-a za datu proceduru. U drugom prolazu kroz listu tokena, otkrivaju se pozivi procedura. Svaki čvor drveta sadrži i listu poziva te procedure. Kada se detektuje poziv, u tu listu se dodaje pointer na čvor stabla koji predstavlja pozvanu proceduru.

Slaganje i forward-ovanje

Ako treba presložiti procedure koje nemaju u sebi deklarisane potprocedure, u memoriji postoji samo koren stabla sa svojim listovima. Tokom slaganja, procedure se ubacuju jedna po jedna, po određenom kriterijumu, u listu složenih procedura. Ta lista na kraju treba da sadrži novi redosled procedura. Njeni elementi su zapravo pointeri koji pokazuju na odgovarajući čvor stabla. Pretražuju se redom svi listovi drveta i ubacuju se u listu oni čvorovi koji sadrže samo pozive procedura koje su već u listi. To znači da na početku, kada je lista prazna, u nju može biti ubačen samo onaj čvor koji se ne referencira ni na jedan drugi čvor. Procedura koja poziva jedino samu sebe, takođe može biti ubačena na početku. I uopšte, prilikom proveravanja se uvek zanemaruju rekurzivni pozivi (procedura je uvek deklarisana za samu sebe). Kada se ispita i poslednja procedura, kreće se iz početka, ukoliko sve procedure nisu ubačene u prethodnom prolasku. Pri ponovnom prolasku, ne proveravaju se procedure koje su već u listi. Ako se dogodi da u jednom prolasku nije ubačena u listu ni jedna procedura, to znači da se negde javlja ,,prsten u pozivima, odnosno uzajamna rekurzija. Za `forward` se u tom slučaju proglašava ona procedura koja učestvuje u najviše ,,prstenova (od onih koje nisu ubačene u listu). Algoritam polazi od prve nerazrešene procedure u nizu i odatle rekurzivno ,,prodire kroz pozive, pamteći u posebnom nizu čvorove koje je ,,posetio . Ako naiđe na neku proceduru koju je već ispitao (koja se vec nalazi u nizu), to znači da sve procedure između dva pojavljivanja te iste procedure učestvuju u jednom ,,prstenu . Njihovi brojači se uvećavaju za jedan, a algoritam odustaje od daljeg ispitivanja tog pravca i vraća se za jedan nivo rekurzije unazad. Kada ispita sve pravce, vratiće se u koren, ispitaće stanja svih brojača, utvrdiće maksimum i odgovarajuću proceduru će proglasiti za forward. Zatim se kontrola ponovo vraća glavnom algoritmu koji započinje novi prolaz kroz procedure. Da bi se razrešio jedan prsten potrebno je forward-ovati samo jednu njegovu proceduru. Znači, da

bi se istovremeno razrešilo najviše prstenova, treba forward-ovati onu proceduru koja učestvuje u najviše njih. Kada su procedure složene, ostaje da se svaka procedura u stablu poveže pointerom sa drugom procedurom u tabeli složenih procedura, koja u konačnom poretku treba da zauzme njeno mesto.

Slaganje celog programa

Obrada složenijeg stabla (sa više od jednog nivoa) vrši se tako što se krene od korena stabla, ali pre nego što se obrade sve procedure iz prvog nivoa, procedura za slaganje rekurzivno pozove samu sebe, kako bi presložila njihove potprocedure. Isti postupak se primenjuje i za same potprocedure. Kao kriterijum izlaska iz rekurzivne procedure služi uslov da je broj listova datog čvora jednak nuli (došlo se do dna stabla). Na taj način, najpre će biti presložene procedure iz najnižeg nivoa stabla, a na kraju procedure glavnog programa (prvi nivo stabla). Postupak se moze prikazati sledećim fragmentom koda:

```
procedure AnalizaStabla(N : NodePtr);
begin
   if not(BrojListova=0) then begin
      for i:=1 to BrojListova do
         AnalizaStabla(N^.List[i]);
      PresloziProcedure;
   end;
end;
```

Svaki put kada se pozove procedura `PresloziProcedure` kreira se posebna lista složenih procedura za taj nivo (a ne za celo stablo). Ovde je bitno da analiza listova određenog čvora (poziv procedure `PresloziProcedure`) prethodi analizi procedura koje su u istom nivou kao i sam čvor. To je posledica činjenice da neke procedure mogu da pozivaju i procedure iz „viših nivoa" stabla. Na primer, u slučaju:

```
procedure Proc2;
   procedure SubProc2;
   begin Proc1; end;
begin ... end;
procedure Proc1;
begin ... end;
```

procedura `SubProc2`, koja je deklarisana u drugom nivou stabla (kao potprocedura procedure `Proc2`), poziva proceduru `Proc1` iz prvog nivoa stabla. To znači da procedura `Proc2` mora u programu da se pojavi pre procedure `Proc1`, iako je neposredno ne poziva. Ovaj problem se rešava tako što se najpre analiziraju potprocedure (`SubProc2`), pa zatim procedure (`Proc1` i `Proc2`). Prilikom analize potprocedure `SubProc2`, poziv će biti prosleđen onoj „roditeljskoj" proceduri koja se nalazi u istom

nivou sa pozvanom procedurom (u ovom slučaju to je `Proc2`) i biće upisan u njenu listu poziva. Tek nakon toga sledi analiziranje procedura `Proc1` i `Proc2`. Algoritam će sada znati da `Proc2` poziva `Proc1` i zameniće im mesta. Program omogućava i `forward`-ovanje modula (unit). Paskal kompajler smatra sve procedure koje su deklarisane u interface delu modula za forward procedure. Zbog toga je potrebno analizirati posebno interface deo još u fazi otkrivanja deklaracija procedura. Sve procedure koje se tu nalaze ubacuju se inicijalno u listu složenih procedura, i ostale procedure ih nadalje mogu pozivati.

Generisanje koda u izlaznoj datoteci i izdavanje poruka o greškama

Kod u izlaznoj datoteci generiše se na taj način što program ponovo prolazi kroz ulaznu datoteku i prepisuje je, osim kada pronađe početak koda procedure. Tada umesto originalne procedure upisuje onu koja treba da je zameni na tom mestu. Ako među ovim procedurama ima onih koje su deklarisane kao `forward`, njihove deklaracije (sa rezervisanom reči forward na kraju), nalaze se ispred svih drugih procedura. Kod unit-a, procedure koje su deklarisane u interface sekciji, ne deklarišu se posebno kao forward, jer se to podrazumeva. Tokom generisanja izlaznog koda, detektuje se svaki upisani CR/LF znak i dopunjavaju se informacije o brojevima linija koje su sakupljene u fazi tokenizacije koda. Sada se pamti i pozicija početka i kraja svake linije u izlaznoj datoteci. Izlazna datoteka se zatim prosleđuje kompajleru, koji vraća poruku o rezultatima kompajliranja. Ona se preusmerava u privremenu datoteku na disku, a u datoteci se pronalaze sva mesta koja se odnose na broj linije u izlaznoj datoteci. Ti brojevi se moraju prevesti na odgovarajuće brojeve linija ulazne datoteke, kako bi čitav proces forward-ovanja i preuređivanja koda bio transparentan za korisnika. Prevođenje se vrši uz pomoć podataka o brojevima linija koji se nalaze u memoriji. Ti podaci su pozicije početka i kraja svake linije koda u ulaznoj i izlaznoj datoteci. Program je prilično neotporan na greške u ulaznom programu, ako se one tiču rezervisanih reči na osnovu kojih program prepoznaje deklaraciju, početak ili kraj procedure. Program će prijaviti grešku „različit broj otvorenih i zatvorenih blokova naredbi , ali pogrešno otkucane reči procedure, function, begin, ili end neće otkriti, već će dati netačan rezultat.

Zaključak

Program je testiran na više desetina primera, od kojih su neki sadržali i oko 100 procedura u jednom fajlu. Potprocedure su bile deklarisane u pet nivoa dubine ili manje. To je veličina koja bi bila dovoljna i za najkom-

plikovanije programe. Sa svim primerima program je radio tačno i brzo (najviše do 2 sekunde za najkomplikovanije primere – nije uračunato vreme kompajliranja), izuzimajući opisane probleme sa sintaksno neispravnim kodom.

Literatura

[1] Jensen, K., Wirth, N. 1989. *Pascal priručnik*. Beograd: Mikro knjiga

[2] Van Wyk, C. J. 1988. *Data Structures and C Programs*. Addison-Wesley

Nikola Milosavljević

Algorithm for Arranging Procedures and Functions of a Pascal Program in an Optimal Order

Very often, while writing a new procedure or function in Pascal code, a programmer has to spend his time looking for a place for it, so that it is declared and ,,visible for all the other procedures that call it. It means that procedure must be declared before all of them, and after the procedures that are called by it. Another problem is mutual recursion. It can be solved by declaring one or more `forward` procedures, but that is only another unconfortability for a programmer. Using this algorithm, these unconfortabilities can be avoided.

Key words: forward procedure, optimal order of procedures, mutual recursion.