PROGRAMIRANJE Z MODULO – 2
1. del

A. Brodnik, M. Špegel, T. Lasbaher
Institut »Jožef Stefan«, Ljubljana

UDK 681.3.06:519.682 MODULO-2

V članku opisujemo programski jezik modula-2. Opis je razdeljen na dva dela. V prvem obravnavamo osnovne podatkovne, ukazne (stavki) in programske (moduli, podprogrami) strukture. Tako je podrobno opisan pojem mudula in z njim povezanega modularnega programiranja. Poleg tega opisujemo tudi osnovna načela sistemskega oziroma nizkonivojskega programiranja v moduli-2. Nadaljevanje članka je namenjeno primerjavi Module-2 s tremi drugimi sodobnimi jeziki, in sicer s pascalom, ado in z jezikom c.

Programming With Modula-2 In this paper is a description of the Modula-2 programming language. The description is divided into two parts. In the first part, we discuss the essential data, control, and programming structures of the language which separate Modula-2 from other modern languages, concluding wise the notation of a module and modular programming, and some principles of system or low-level programming. In the second part of this contribution, Modula-2 is compared to other three modern programming languages: Pascal, Ada and C.

1. UVOD

V članku želimo v dveh delih predstaviti nov, moderen programski jezik. Za celovit opis jezika pa je okvir tega članka in njegovega nadaljevanja preozek, poleg tega pa je bralcu na voljo vrsta dobrih knjig [3, 7, 8, 14, 18]. Pač pa je namen našega prispevka z opisom bistvenih novosti in programirnih možnosti vzpodbuditi bralca k poglobljenemu spoznavanju tega nedvomno zelo sposobnega in morda enega temeljnih jezikov naslednjih let [2].

Programski jezik modula-2 je neposredni naslednik module-1, oba pa izvirata iz pascala. Osnovnim pravilom so bili dodani predvsem nekateri elementi paralelnega pascala ("Concurrent Pascal")[5] ter možnost modularnega programiranja. Zato osnovo jezika tvorijo že dobro poznani stavki iz pascala [6, 9], ki pa so v moduli-2 še bolj poenostavljeni. S tem je jezik postal preglednejši, programi pa bistveno bolj zgoščeni. Modula-2 ni boljša od pascala samo v pogledu strukturiranosti, ampak omogoča z določenimi ukaznimi strukturami tudi zelo učinkovito programiranje na sistemskem nivoju.

Izkušeni programerji, ki sicer uporabljajo katero od bolj izpopolnjenih različic pascala [15, 16], bodo dejali, da vse to, kar sicer prinaša modula-2, oni lahko že precej časa uporabljajo. To je res, vendar se moramo hkrati zavedati, da so to le dialekti pascala, medtem ko so vsi deli jezika, ki jih bomo tu opisali, že v osnovni definiciji module-2. Opis jezika je osnovna tema drugega poglavja. V tretjem poglavju, bomo prikazali nekatera načela modularnega programiranja, četrto poglavje pa je namenjeno prikazu zmožnosti module-2 za sistemsko ali nizkonivojsko programiranje.

V drugem delu tega članka bomo primerjali jezik s pascalom, ado in jezikom c. Pri tem se bomo seznanili z nekaterimi prav zanimivimi spoznanji. Videli bomo na primer, da ima modula-2 pravzaprav skoraj vse bistvene prednosti, ki jih programerji sicer pripisujejo adi in c-ju.

V drugem poglavju nadaljevanja so zbrani še primeri nekaterih prevajalnikov za različne operacijske sisteme in zanimiv primer povezave module-2 s prologom.

V dodatku sta priložena kratka primera, ki naj predočita možnosti modularnega in sistemskega programiranja v moduli-2.

2. Opis jezika modula-2

Pri razmišljanju o uporabi kakšnega programskega jezika, moramo vzeti v pretres predvsem tri gradnike: podatkovne strukture, ukazne strukture (stavke) in programske strukture. Opis jezika smo zato razdelili nekako na tri dele.

Zaradi smotrne zgradbe jezika so programi napisani v moduli-2 [18] zelo pregledni. Jezik dobro podpira strukturiranje programov, in sicer tako v pogledu podatkovnih kot ukaznih struktur. Razen tega modula-2 loči med velikimi in malimi črkami v imenih spremenljivk. Preglednost programov povečuje še zahteva, da so vse rezervirane besede napisane z velikimi črkami.

Pri opisu jezika predpostavljamo določeno stopnjo znanja katerega od algolskih jezikov, a najbolje je poznavanje pascala.

2.1 Podatkovne strukture

Vsak programski jezik pozna nekaj osnovnih tipov. Najbolj običajni osnovni tipi so **integer** (cela števila), **real** (realna števila) in **boolean** (logične spremenljivke) in jih najdemo v večini jezikov od fortrana [12] pa do ade [13, 17]. V večini jezikov algolskega tipa, kamor sodi tudi modula-2, poznamo še tipe **char** (črkovna spremenljivka), **set** (množica) in **pointer** (kazalec).

V modulo-2 so vgrajeni vsi zgoraj navedeni tipi, poleg njih pa pozna jezik še nekaj tipov, ki so uporabni predvsem pri programiranju na

nižjem nivoju. To so tipi **CARDINAL** (nenegativna števila), **WORD** (beseda), **ADDRESS** (naslov), **BITSET** (množica bitov) in **PROC** (podprogram). Vlogo teh tipov si bomo ogledali v četrtem poglavju tega prispevka.

2.1.1 Sestavljeni tipi

Podobno kot pascal pozna tudi modula-2 sestavljene podatkovne tipe. To so zapisi (RECORD), zapisi z različicami, polja (ARRAY), naštevalni tipi in delni tipi. Oglejmo si oblike definicij za omenjene podatkovne tipe na naslednjem primeru:

```
TYPE
  zapis  = RECORD part1, part2: CARDINAL END;
  zapis1 = RECORD
             CASE selekt: BOOLEAN OF
               TRUE:  ch: CHAR  |
               FALSE: ca: CARDINAL
             END
           END;
  polje  = ARRAY [1..10], [1..10] OF INTEGER;
  nastej = (da, ne, mogoce);
  delni  = [0..9];
```

2.1.2 Pretvarjanje med tipi

V moduli-2 je obravnavanje spremenljivk močno vezano na njihov tip. Ne moremo na primer, seštevati celega in realnega števila kot v pascalu. Prepovedano je celo seštevanje celega in nenegativnega števila. Za premostitev teh problemov obstaja cela vrsta funkcij, ki samo na logičnem nivoju pretvorijo tip spremenljivke, na velikost izhodne kode pa nimajo vpliva. Imena teh funkcij so enaka imenom tipov. Tako lahko naredimo naslednjo pretvorbo:

```
MODULE Primer1;
VAR c: CARDINAL;
    i: INTEGER;
BEGIN
  i := INTEGER (c); c := CARDINAL (i)
END Primer1.
```

Seveda je pretvarjanje omejeno le med tipi z enako dolžino. Kot primer pravilne pretvorbe, si oglejmo naslednji podprogram:

```
PROCEDURE Prepisi (iz: IzTip;
                   VAR v: vTip;
                   VAR prepisano: BOOLEAN);
BEGIN
  prepisano := TSIZE(IzTip) = TSIZE(vTip);
  IF prepisano THEN v := vTip (iz) END
END Prepisi;
```

Klicani podprogram TSIZE je eden redkih, ki so definirani v sistemu module-2 in izračuna velikost tipa, ki je parameter podprograma, v besedah.

2.2 Ukazne strukture

Modula-2 pozna naslednje stavke:
- prirejanje
- klic podprograma
- vrnitev iz podprograma (RETURN)
- pogojni stavek (IF, ELSIF, ELSE)
- izbiralni stavek (CASE)
- izhodni stavek (EXIT)
- strukturni izbirni stavek (WITH)
- zanke
  - nedoločna (LOOP)
  - pogoj na začetku (WHILE)
  - pogoj na koncu (REPEAT)
  - naštevalna zanka (FOR)

Modula-2 torej ne uvaja nobenih bistveno novih stavkov, pač pa je le njihova oblika lepše zapisana. Na primer, modula-2 ne pozna sestavljenega stavka (BEGIN ... END), ampak se vsi njeni stavki razen prirejanja, vrnitvenega in izhodnega končajo z rezervirano besedo END. V primerjavi s pascalom lahko opazimo odvečnost besedice **begin** na naslednjem primeru:

```
MODULE Primer;          program Primer;
BEGIN                   begin
  IF pogoj THEN           if pogoj then begin
    stavek1;                stavek1;
    stavek2                 stavke2
  END                     end
END Primer.             end.
```

Novo je tudi ločevanje posameznih možnosti v izbiralnem stavku. Ta ima sedaj obliko:

```
CASE izraz OF
  moznost1 : Stavki1 |
  moznost2 : Stavki2
  ELSE Stavki3
END;
```

2.3 Moduli in podprogrami

Osnovna programska enota v jeziku ni več podprogram, temveč modul. Ta lahko združuje več podprogramov in modulov. Modul in podprogram se razlikujeta tudi v svoji zasnovi, saj je modul le logična, programska enota, ki obstaja takorekoč samo na nivoju izvorne kode in je njegova vloga samo povečanje preglednosti programa. Po drugi strani pa podprogram predstavlja osnovno izvajalno enoto in obstaja tako na nivoju izvorne kot tudi izvajane kode.

2.3.1 Moduli

Modula-2 poznam tri tipe modulov:
- programski modul
- definicijski modul (DEFINITION)
- delovni modul (IMPLEMENTATION)

Prvi služi kot osnovni uporabnikov program ali kot ga poznamo v pascalu **program**. Druga dva tipa modulov sta dodana predvsem zaradi zaokrožitve sistema jezika, s čimer se odpravijo različne uporabniške knjižnice. Modula-2 ne pozna več zunanjih podprogramov (**external**) ali vstavljanja celih kosov programa (**include**), ampak samo navedemo modul, iz katerega želimo uporabiti določeno spremenljivko (ta je lahko poljubnega tipa, tudi PROC!).

Uporabnost definicijskih in delovnih modulov je najbolj očitna pri snovanju velikih projektov, pri katerih vedno sodeluje več ljudi. Takšno programiranje zahteva razdelitev dela in določitev nekaterih skupnih točk - spremenljivk. V primeru razvoja z modulo-2 so spremenljivke opisane z definicijskimi moduli. V definicijskih modulih torej snovalci opišejo kaj bodo podprogrami delali, kakšne bodo posamezne spremenljivke in podobno. Nato se programerji lotijo vsak svojega dela, kar pomeni, da vsak izdela svoj delovni modul, v katerem se določi **kako** se bodo izvajali.

Opisani postopek izdelave projekta v osnovi omogoča zelo enostavno popravljanje programov, saj ni potrebno prevesti zaradi popraviti ničesar drugega kot modul, ki je napačen, seveda ni pa edini možen način uporabe ločenega prevajanja posameznih modulov. Sicer ločeno prevajanje omogočajo že običajne sistemske knjižnice, ki pri drugih jezikih obstajajo izven opisa jezika, medtem ko je pri moduli-2 vse vgrajeno v sistem jezika samega. Podrobneje si bomo ogledali načela modularnega programiranja v tretjem poglavju.

Naslednja posebnost modulov je veljavnost\spremenljivk. Oglejmo si, kako v moduli-2 dostopamo do spremenljivk. Osnovno programsko enoto -

modul si lahko predstavljamo kot zaprt prostor, v katerem nam je vse vidno, zato pa ne vidimo ničesar zunaj tega prostora. Če želimo videti kakšno stvar, ki je zunaj prostora, si jo moramo posebej dodefinirati: sprejeti jo moramo v modul. V moduli-2 to naredimo s stavkom IMPORT. Ne moremo pa sprejemati poljubnih spremenljivk, ampak samo tiste, ki so na voljo v zunanjem svetu. Da lahko neka spremenljivka obstaja v zunanjem svetu, jo moramo iz modula oddati. Za to služi stavek EXPORT. Torej si lahko vse oddane spremenljivke predstavljamo kot nekakšno prosto blago, ki ga lahko sprejme katerikoli modul.

To načelo modulov je tako močno, da tudi vsebina vsebovanih modulov ni poznana osnovnemu modulu. Da bo stvar jasnejša si poglejmo naslednji primer:

```
MODULE Ena;
  VAR x, y, z: CARDINAL
  MODULE Dva1;
    EXPORT a, b;
    VAR a, b, c: CARDINAL;
  BEGIN
    (* tu so dostopne
       spremenljivke a, b ali c *)
  END Dva1;

  MODULE Dva2;
    IMPORT x, a;
    VAR r, s, t: CARDINAL;
  BEGIN
    (* dostopne so spremenljivke
       x ali Ena.x, a ali Dva1.a,
       r, s, t *)
  END Dva2;
BEGIN
  (* tu pa spremenljivke x, y, z
     in Dva1.a ali a *)
END Ena.
```

2.3.2 Podprogrami

Poleg modulov modula-2 še vedno pozna tudi podprograme. Njihova definicija je podobna kot pri pascalu, adi ali drugih jezikih. Parametri podprogramov so lahko klicani po vrednosti ali po naslovu (referenci). V podprogramu nekega modula so vidne tudi vse zunanje spremenljivke, ki pa so definirane v tem modulu. Dostop do ostalih spremenljivk je možen preko oddajno/vnosnih struktur. Za lažjo predstavitev naj služi naslednji primer:

```
MODULE Glavni;
  VAR c: CARDINAL;
  MODULE Ena1;
    EXPORT a;
    VAR a: INTEGER;
  END Ena1;
  MODULE Ena2;
    IMPORT a, c;
    EXPORT Priredi;
    PROCEDURE Priredi;
    BEGIN
      a:=INTEGER(c)
    END Priredi;
  END Ena2;
BEGIN
  Priredi;
END Glavni.
```

V podprogramu lahko nastopa tudi vrnitveni stavek (RETURN), ki pomeni takojšen zaključek izvajanja podprograma. Če tega stavka ni, se vrnitev izvede, ko izvajanje podprogram pride do konca (END). V primeru funkcijskega podprograma sledi vrnitvenemu stavku izraz, ki določa vrednost funkcije.

Definicija funkcijskih podprogramov je pri moduli-2 in pascalu različna, saj jih pri pascalu označuje rezervirana beseda function. V moduli-2 funkcijo ločimo od podprograma le po tem, da je v glavi definicije naveden tip. Primer:

```
PROCEDURE PraviPodprogram;
BEGIN
END PraviPodprogram;

PROCEDURE Funkcija: tip;
BEGIN
END Funkcija;
```

Pri definiciji tipov podprogramskih parametrov obstajajo nekatere možnosti sproščanja sicer trdnega sovpadanja tipov. In sicer lahko uporabimo splošne tipe:

- WORD: osnovna računalniška beseda, sicer posplošenje kateregakoli tipa, ki je dolg eno besedo
- ADDRESS: kazalec na poljuben tip, sicer je definiran kot
  ADDRESS = POINTER TO WORD;
- BITSET: poljubna množica, sicer množica 16 bitov
- ARRAY OF tip: odprto polje poljubnega tipa, katerega velikost lahko določimo s funkcijo HIGH

Sistem jezika pozna zelo malo standardnih funkcij in podprogramov. S tem se je jezik izognil odvisnosti od določenega operacijskega sistema ali računalnika. Vsi standardni podprogrami izvirajo iz definicije jezika in omogočajo predvsem bolj parametrizirano pisanje programov. Seznam standardnih podprogramov si lahko bralec ogleda v knjigi [18].

Na tem mestu naj posebej poudarimo, da podprogrami za branje z različnih enot oziroma datotek ostajajo povsem uporabnikova skrb. Vendar proizvajalci prevajalnikov vedno dodajajo sistemu jezika nekakšno standardno množico modulov, ki omogočajo bralne in pisalne operacije na različne enote ter ostalo osnovno delo z operacijskim sistemom. Hkrati se v svetu vedno bolj čuti potreba po standardizaciji teh modulov. Določen naj bi bil predvsem vmesnik med modulo-2 in vhodno-izhodnimi funkcijami, ki so odvisne od posameznega sistema. Delovno ime tega projekta je OSSI in pomeni Operating System Standard Interface [1]. Izvaja se na večih mestih, njegovo središče pa je v Zurichu.

2.4 Procesi

Kot nasledstvo concurrent pascala so procesi. Proces predstavlja zaporedje ukazov, ki se izvajajo neprekinjno. Proces je tudi nekako osnovna fizično izvajana enota. Modula-2 omogoča uporabo sorutin (coroutines), kar pomeni, da nima vgrajenega razporejevalnika. Proces lahko poženemo s podprogramom TRANSFER, ki preda nadzor od trenutnega procesa naslednjemu, ki je naveden kot parameter. Podrobnejši opis tega mehanizma sledi v četrtem poglavju.

3. Modularno programiranje

Dandanes, ko cena strojne opreme vztrajno pada in se cena programske opreme dviguje, postajajo načela učinkovitega programiranja vse bolj pomembna. Prav modularno programiranje se je izkazalo za eno od temeljnih načel. V ZDA so v sedemdesetih letih sestavili celo ekipo strokovnjakov, ki naj predložila nov jezik, ki bo omogočal hitro in učinkovito, torej modularno programiranje. Nastala je ada. Vendar se je kmalu izkazalo, da je jezik prevelik, zaradi česar nikakor ni mogel dobiti širšega področja uporabe. Hkrati z njim se je pojavil manjši, vendar še vedno zelo učinkovit jezik - modula-2.

3.1 Delitev na module

Sedaj, ko smo dobili možnost uporabe ločenih modulov, jih moramo še nekako pametno uporabiti. Večina programerjev si vedno postavlja vprašanje: "Kaj sodi v en modul?".

Odgovor, ki bi bil plod globljega premisleka, bi se naslonil na dejstvo, da so pri izdelavi kakršnegakoli projekta podatki in njihove strukture navadno pomembnejše kot nadzorne strukture. Torej vežemo modul na pojem podatkovne strukture. Pomberger [11] sicer navaja še več načel pri delitvi programa na module, vendar na konou vseeno ugotavlja, da so podatkovne strukture najpomembnejši dejavnik, ki določa vsebino modula. Zmeraj pa to ni res. Kot protiprimer so moduli, ki upravljajo različne enote (driver, handler). V drugem delu članka bomo v enem od primerov predstavili takšen modul.

3.2 Kako so moduli povezani

Seveda mora modul vseeno še ohranjati stik z okolico. V ta namen nam služijo nekatere spremenljivke, predvsem pa različni podprogrami. Ker so podprogrami del modula, je s tem zagotovljena varnost podatkov pred napačno uporabo. V moduli-2 dosežemo povezavo z oddajo in vnosom določenih spremenljivk (EXPORT in IMPORT stavek). Naj zopet poudarimo, da je spremenljivka lahko poljubnega tipa in torej lahko tudi tipa PROC! Okolica seveda natančno pozna tipe oddanih spremenljivk in tudi parametrov v oddanih podprogramih že v času prevajanja. Prav močna zahteva po sovpadanju tipov je eno od osnovnih načel module-2. S tem so tudi odpravljene tako pogoste pomote, ki sicer rade nastopajo v drugih jezikih (fortran, pl/1 in drugi).

Za primer, kaj bi povezovalo nek modul z zunanjostjo, si oglejmo modul, ki hrani podatkovno strukturo sklada. Smiselno je oddati le podprograma Vstavi in Vzemi ter funkciji Prazen in Poln. To v moduli-2 definiramo na naslednji način:

```
DEFINITION MODULE NadzornikSklada;

  EXPORT Vstavi, Vzemi, Prazen, Poln;

  PROCEDURE Vstavi (el: CARDINAL);
  PROCEDURE Vzemi (VAR el: CARDINAL);
  PROCEDURE Prazen (): BOOLEAN;
  PROCEDURE Poln (): BOOLEAN;
END NadzornikSklada.
```

To je za nekoga, ki želi uporabljati sklad povsem dovolj. Kako programer deluje nad skladom, za njega ni važno.

3.3 Ločeno prevajanje

S tem, da smo razbili program na module, nismo le zaščitili posameznih podatkovnih struktur ali dostopov do različnih enot, ampak smo pridobili tudi možnost, da lahko vsak modul prevajamo povsem neodvisno od ostalih. Kot dokaj preprost a zelo pomemben primer uporabnosti ločenega prevajanja [4] je vzdrževanje programske opreme. Recimo, da se v obdobju izkoriščanja nekega programskega paketa izkaže, da je sklad, ki ga uporabljamo preko modula NadzornikSklada premajhen. Vse kar je treba spremeniti, je delovni modul, ga prevesti in ponovno vse skupaj povezati. Ostali moduli se pri tem prav nič ne spremenijo!

Naj na koncu poglavja navedemo še delovni modul NadzornikSklada, kot bi ga napisal povprečen programer. Če pa boste imeli probleme z velikostjo sklada, povečajte konstanto StackLimit. V primeru, da vam tudi to ni dovolj, lahko napišete nov modul, ki bo sklad hranil na datoteki. Vendar pozor! Ostali programi se tudi v tem primeru ne bodo prav nič spremenili!

```
IMPLEMENTATION MODULE NadzornikSklada;

  CONST VelikostSklada = 1000;
  VAR sklad: ARRAY [1..VelikostSklada]
               OF CARDINAL;
      n: CARDINAL;

  PROCEDURE Vstavi (el: CARDINAL);
  BEGIN
    IF n < VelikostSklada THEN
      n:=n+1; sklad[n] := el
    END
  END Vstavi;

  PROCEDURE Vzemi (VAR el: CARDINAL);
  BEGIN
    IF n > 0 THEN
      el := sklad[n]; n:=n-1
    END
  END Vzemi;

  PROCEDURE Prazen (): BOOLEAN;
  BEGIN
    RETURN n = 0
  END Prazen;

  PROCEDURE Poln (): BOOLEAN;
  BEGIN
    RETURN n = VelikostSklada
  END Poln;

BEGIN  (* inicializacija modula *)
  n := 0
END NadzornikSklada.
```

4. Sistemsko programiranje

Možnost sistemskega ali nizkonivojskega programiranja (low level programming) je ena bistvenih postavk, ki jih mora omogočati sodoben programski jezik. Vsekakor mora biti možen dostop do poljubnih naslovov v pomnilniku in pisanje prekinitvenih podprogramov. Prav slednja zahteva je močno povezana s pojmom procesa in še bolj s pojmom paralelnih procesov. Modula-2 je v pogledu zmožnosti opisovanja paralelnih procesov zelo močan jezik. Že v osnovi pozna pojem procesa. Procesi se izvajajo kot sorutine (coroutines), vendar lahko uporabnik ob ustrezni zunanji prekinitvi napiše razporejevalnik (schedular), ki omogoča navidezno sočasno izvajanje večih procesov. V drugem delu članka bomo med primeri prikazali prav takšen razporejevalnik.

4.1 Dostop do pomnilnika

Modula-2 omogoča dostop do poljubnega mesta v pomnilniku na dinamični ali statični način. Statični način (podobno kot pri OMSI pascalu [10]) pomeni, da ob definiciji spremenljivke navedemo tudi mesto, kjer se nahaja.

```
VAR NaNaslovu10 [10]: CARDINAL;
```

Dinamičen dostop pa je možen preko splošnega tipa ADDRESS, ki je definiran v modulu SYSTEM in ima obliko:

```
TYPE ADDRESS = POINTER TO WORD;
```

Tako lahko uporabljamo opisano mesto v pomnilniku tudi na sledeč način:

```
MODULE Primer2;
  FROM SYSTEM IMPORT ADDRESS;
  CONST xxx = 0;
  VAR a: ADDRESS;
BEGIN
  a := ADDRESS (10);
  a^ := xxx;          (* NaNaslovu10 *)
END Primer2.
```

Sicer pa lahko definiramo tudi celoten, recimo 16-biten pomnilnik kot polje na naslednji način:

```
CONST MaxNaslov = 177777B;
VAR   Pomnilnik [0B]:
        ARRAY [0..MaxNaslov] OF WORD;
```

4.2 Procesi in prekinitveni podprogrami

Za lažje razumevanje možnosti pisanja prekinitvenih podprogramov v moduli-2 si najprej oglejmo, kako jezik uporablja procese.

Proces je definiran kot osnovna izvajalna enota, ki se izvaja zaporedno (sekvenčno). Seveda lahko nek proces proži in ustvarja druge procese. V moduli-2 je proces poseben tip, ki ga oddaja modul SYSTEM. V dodatku k definicijam jezika [18] je ta tip sicer opuščen, vendar ga bomo v naših primerih zaradi preglednosti še uporabljali. Namesto tipa PROCESS se sedaj uporablja enostaven tip ADDRESS. Proces se izvaja zaporedno in neprekinjeno, dokler ne požene nekega drugega procesa. To izvede s podprogramom

```
PROCEDURE TRANSFER (VAR od, komu: PROCESS);
```

ki ga oddaja zopet modul SYSTEM. Ta modul tudi oddaja podprogram

```
PROCEDURE NEWPROCESS (p: PROC; a: ADDRESS;
                      n: CARDINAL;
                      VAR pr: PROCESS);
```

kjer je p podprogram, ki se bo izvajal kot samostojen proces. Svoje spremenljivke (sklad) bo imel v prostoru, ki se prične na mestu a in je velik n besed. Vse potrebne podatke o novem procesu shrani podprogram v parameter pr.

Naslednji korak je povsem razumljiv in sicer definiramo podprogram, ki poslužuje neko prekinitev kot nov proces. Problem je le še, ker ga je treba v pravem trenutku pognati s podprogramom TRANSFER. V ta namen modul SYSTEM oddaja še en podprogram, katerega definicijo bomo napisali sedaj za računalnike, ki imajo prekinitve vektorirane.

```
PROCEDURE IOTRANSFER (VAR komu, od: PROCESS;
                      vektor: CARDINAL);
```

V bistvu sistem ob kliou tega podprograma poveže proces od z vektorjem vektor in požene proces komu. Ta se izvaja, dokler se ne zgodi prekinitev in takrat se zopet požene proces od. Tako dobimo skelet prekinitvenega podprograma:

```
PROCEDURE Skelet;
BEGIN
  Inicializacija;
  LOOP
    IOTRANSFER (program, posluzi, vector);
    PosluziPrekinitev
  END
END Skelet;
```

Primer takšnega podprograma bomo navedli v naših primerih, ki bodo navedeni v drugem delu članka.

5. Zaključek

Na tem mestu zaključujemo prvi del našega prispevka. V njem smo želeli podati osnovne zmožnosti programskega jezika modula-2. V skopih besedah smo opisali osnovne podatkovne, ukazne in programske strukture jezika.

V nadaljevanju smo opisali osnovna načela modernega načina programiranja, za katerega se je v svetu udomačil izraz modularno programiranje. Zadnje poglavje prvega dela opisuje možnosti nizkonivojskega in sistemskega programiranja v moduli-2.

Zahvala

Za vso pomoč pri snovanju besedila in dragocene nasvete pri iskanju primernih slovenskih izrazov za angleške besede se najlepše zahvaljujem prof.dr.Boštjanu Vilfanu.

Literatura

[1] E.Biagioni, K.Hinrich, G.Heiser, C.Muller: A Portable Operating System Interface and Utility Library, IEEE Software, November 1986.

[2] R.P.Cook: Modula-2 Experiments Will Help Future Language Design, IEEE Software, November 1986.

[3] R.Gleaves: Modula-2 for Pascal Programmers, Springer-Verlag, 1984.

[4] J.Gutknecht: Separate Compilation in Modula-2: An Approach to Efficient Symbol Files, IEEE Software, November 1986.

[5] P.B.Hansen: The programming language Concurrent Pascal, IEEE Trans. on Software Engineering SE-1, 2 (junij 1975), 199-207.

[6] K.Jensen, N.Wirth: PASCAL User Manual and Report, Springer-Vgrlag, 1975.

[7] Modula-2 Development System, Modula Corporation Provo, Utah< 1984.

[8] Modula-2 System for Z80 CP/M, Hochstrasser Computing AG, Zurich, Switzerlind, 1984.

[9] B.Mohar, E.Zakrajšek: Uvod v programiranje, DMFA, Ljubljana, 1982.

[10] OMSI Pascal-1, version 1.2 Manual for RT-15, Oregon Software, Oregon, 1981.

[11] G.Pomberger: Software Engineering and Modula-2, Prentice-Hall, 1984.

[12] Programski jezik Fortran, Iskra Delta, Ljubljana, 1982.

[13] I.C.Pyle: The Ada Programming Language, Prentice-Hall, 1981.

[14] SDS-XP, Extended Performance Modula-2 Software Development System, Ynterface Technologies, Houston, Texas, 1985.

[15] TURBO PASCAL, Version 3.0, Reference Manual (1985), Borland International Inc., 4585 Scotts Valley Drive, Scotts Valley, CA 95066.

[16] VAX-11 Pascal Language Reference Manual, Digital Equipment Corporation, Maynard, Massachusets.

UDK 681.3.06 MODULA – 2

Andrej Brodnik, Marjan Špegel, Tadej Lasbaher
Institut »Jožef Stefan«, Ljubljana

Ta prispevek obsega primerjavo module-2 s programskimi jeziki pascal, ada in c, splošen opis prevajalnika za modulo-2 in opis najbolj vidnih prevajalnikov za različne operacijske sisteme. Sklepni del prispevka je namenjen kratkemu opisu zanimive združitve zmožnosti funkcionalnega in proceduralnega jezika (module-2 in prologa). V dodatku navajamo dva primera programov v moduli-2, ki naj bi predstavila tako osnovna načela sistemskega programiranja, kot postopek modularnega programiranja. Ta prispevek je nadaljevanje prej objavljenega članka in zaokrožuje kratko predstavitev jezika modula-2.

Programming With Modula-2 In this paper the comparisson of Modula-2 with Pascal, Ada and C is given. Further, the general overview of compiler is presented with some specific examples for several operating systems. Finally, the very smart connection of functional and procedural language (Prolog and Modula-2) is given. The appendices contain two examples of programmes which should give a flavour of modular and system programming in Modula-2. This paper with prevecously published one closes the sketching of Modula-2 programming language.

1. UVOD

Če je bil prvi del prispevka namenjen jeziku samemu, potem poskušamo v tem delu jezik postaviti nekam v okolje. Tako navajamo nekatere podatke, ki bodo pomagale oceniti razmerje module-2 do drugih programskih jezikov. Temu so namenjena prva tri poglavja.

V nadaljevanju opisujemo sam princip prevajanja programov napisanih v moduli-2 in nivojski ustroj splošnega prevajalnika. V zaključnih dveh poglavjih pa poskušamo prikazati nekaj zelo zanimivih primerov uporabe jezika na dokaj različnih primerih zunaj in pri nas.

V posebnem dodatku vam predstavljamo dve lupini programov, ki sta napisana v moduli-2 in sicer razporejevalnik procesorskega časa in program za pogovor med dvema terminaloma.

2. PASCAL IN MODULA-2

Že v prejšnjem sestavku o moduli-2 smo opozorili na to, da je modula-2 kakovostna nadgraditev pascala. Po njem je podedovala predvsem bločno zgradbo in vse osnovne tipe stavkov, od njega pa se bistveno razlikuje predvsem v naslednjih pogledih [5]:

a) gradnik "modul" omogoča
   - ločeno prevajanje programskih in podprogramskih modulov
   - enostavno gradnjo modulov s programskimi orodji
   - knjižnice, ki so podprte s sistemom jezika

b) nizkonivojsko ali sistemsko programiranje
   - dostop do poljubnega stalnega mesta v pomnilniku
   - dostop do posameznih procesorskih registrov
   - pisanje prekinitvenih podprogramov
   - pisanje splošnih sorutin

c) podprogram je definiran kot poseben tip

d) sintaktične razlike

Razlike pod točkami a, b i c so bistvene in so semantične narave, medtem ko so razlike pod točko d predvsem oblikovne oziroma sintaktične narave. Ker pascal ne pozna pojma modula in ker ne omogoča sistemskega programiranja, ga na teh dveh področjih tudi ne moremo primerjati z modulo-2.

2.1 Podprogram kot tip

Podprogram kot tip je v zelo omejenem smislu sicer poznal že pascal in sicer kot parameter podprograma [6, 16], ki je moral biti brez parametrov. V moduli-2 je povsem pravilna definicija naslednjega tipa:

```
TYPE
  ProcType = PROCEDURE (Type1, Type2): Type3;
```

Parametri podprogramov so prav tako lahko kakršnegakoli tipa.

2.2 Sintaktične razlike

Modula-2 je podobna pascalu tudi po zunanjem videzu. Vendar vseeno obstajajo nekatere razlike med obema jezikoma in jih navajamo tukaj:

a) modula-2 loči med velikimi in malimi črkami. Na primer: **ena** ni ista spremenljivka kot **ENA**
b) pascal pozna dva tipa komentarjev, medtem ko je v moduli-2 dovoljen samo eden, ki pa ga lahko gnezdimo. Tako v moduli-2 { to ni komentar }, (* ampak to *), (* ki je lahko (* gnezden *) *).
c) modula-2 pozna več osnovnih tipov kot pascal.
d) v moduli-2 so dovoljene različne okrajšave za veččrkovne operatorje. Znak neenakosti <> se zapiše kot #; logični **AND** je dovoljeno okrajšati v &.

e) modula-2 ne pozna sestavljenega stavka, kajti izkaže se, da je nepotreben. Tako je besedica **begin** v naslednjem primeru povsem odvečna:

```
pascal:
  for i:=1 to 10 do
    begin
      stavek1;
      stavek2;
       .  .  .
      stavekn
    end;

modula-2:
  FOR i:=1 TO 10 DO
    stavek1;
    stavek2;
     .  .  .
    stavekn
  END;
```

f) modula-2 pozna brezpogojno zanko (LOOP stavek).
g) stavka IF c1 THEN s1 ELSIF c2 THEN s2 ELSE s3 END; pascal ne pozna.
h) v moduli-2 se vsi stavki zaključijo z rezervirano besedo END, kar je razvidno že iz primera za sestavljene stavke.
i) v moduli-2 je povsem prepovedan **goto** stavek.
j) razlika je pri definiciji funkcijskih podprogramov.
k) modula-2 ne pozna standardnih podrogramov get, put, read, readln, write, writeln za delo z vhodno izhodnimi enotami.
l) modula-2 ne pozna tipa **file**.

Tu naštete razlike seveda niso edine, vendar so po našem mnenju najpomembnejše.

3. ADA IN MODULA-2

Ada [21, 24] in modula-2 sta po letnici rojstva vrstnici, zato ni nič čudnega, če v obeh jezikih najdemo nekatere bistvene podobnosti, kljub bistvenim sintaktičnim, zlasti pa semantičnim razlikam. Zelo pomembna razlika je že kar velikost jezika. Kot navaja Pomberger [20], obsega formalna definicija ade več sto strani, medtem ko za modulo-2 zadostuje že dobrih 25. Temu primerna je tudi razlika v velikosti prevajalnikov in kode, ki jo naredi prevajalnik. Zato na majhnih računalnikih popolnih prevajalnikov za ado praktično ni, medtem ko jih za modulo-2 najdemo kar nekaj [9, 22].

Podobnosti med ado in modulo-2 si oglejmo na primerjavi podatkovnih in ukaznih struktur, na primerjavi modularnosti in možnosti ločenega programiranja ter na primerjavi sposobnosti jezikov za nizkonivojsko programiranje in za vzporedno izvajanje postopkov.

3.1 Podobnost podatkovnih in ukaznih struktur

Ada in modula-2 poznata podobne osnovne podatkovne tipe, le da ada obvlada tudi števila v takoimenivani stalni vejici. V moduli-2 realiziramo ta števila s sestavljenim tipom:

```
FixedPoint = RECORD
               num: CARDINAL;
               base, delta: REAL
             END;
```

Razen tega ada pozna podtipe, ki jih v moduli-2 lahko vodimo kot delne tipe, po drugi strani pa ada ne pozna tipa PROC. Poleg tega modula-2 pozna tip množice, ki ga v adi realiziramo s tabelo:

```
BITSET is array (0..15) of BOOLEAN;
```

in podatkovni tip ADDRESS, ki ga lahko v adi definiramo kot

```
ADDRESS is access WORD;
```

Razlike med ukaznimi strukturami jezikov so bistveno manjše. Oba jezika poznata podobne osnovne stavke, vendar ima ada pri tem nekaj več možnosti. Modula-2, na primer, ne pozna stavka **declare** (nadomestimo ga lahko z uporabo novega modula) in **delay** stavka (nadomestimo ga s klicem ustreznega podprograma). Modula-2 tudi povsem prepoveduje uporabo GOTO stavka, ki pa je v adi dovoljen. Naslednji stavek, ki ga najdemo v adi in ne v moduli-2, je **null** stavek. V resnici obstaja tudi v moduli-2, vendar ga tam ne napišemo, saj je predstavljen s praznim stavkom, ki ima obliko samo podpičja ali celo ničesar. Kako v moduli-2 nadomestimo **raise** stavek iz ade, pa bomo opisali na primeru nadomeščanja strukture **exception** in njene uporabe.

Ada je tudi precej močnejša kot modula-2 pri uporabi podprogramov, vendar lahko prednosti ade dosežemo s pametnim programiranjem tudi v moduli-2. Tako adino prednost, da je rezultat funkcijskega podprograma lahko tudi strukturiran tip, v moduli-2 nadomestimo s tem, da izračunani rezultat vrnemo preko parametra, ki je bil klican po naslovu (referenci). Pač pa v moduli-2 ne obstaja možnost ponovnega definiranja osnovnih operacij, kot so na primer množenje, seštevanje in podobno (infiksne operacije), in si moramo zato pomagati s klici ustreznih podprogramov.

3.2 Posebnosti in osnovni podprogrami

V adi obstajata dve strukturi, ki jih modula-2 nima: prva je **exceptions** (posebnosti) in je namenjena predvsem za obravnavanje posebnih stanj (na primer napak) v izvajanju programa. Te strukture v moduli-2 v bistvu sploh ne potrebujemo; če smo pravilno razbili naš program, moduli sami prestrežejo vse napake in jih ostali program sploh ne zazna.

Druga struktura, ki jo najdemo samo v adi, izhaja iz dejstva, da je nek algoritem (recimo za vodenje sklada) povsem neodvisen od tega, kakšni elementi so v skladu. V ta namen pozna ada takoimenovane osnovne podprograme (**generic**), kar je v moduli-2 še najtežje nadomestiti. Pri tem si pomagamo z splošnimi tipi (ADDRESS, WORD, BITSET, ARRAY) in predvsem z možnostjo ločenega prevajanja, kjer uporabimo za dano različico dani modul.

3.3 Modularnost in ločeno prevajanje

Načela modularnosti in možnosti ločenega prevajanja smo si že ogledali v četrtem poglavju prvega dela našega prispevka. Na tem mestu pa si oglejmo le podobne strukture ade.

Modularno programiranje in ločeno prevajanje omogoča ada enako dobro kot modula-2. Pojem modula nadomešča v adi pojem paketa (**package**). Kot modula-2 tudi ada loči med definicijskim in delovnim modulom. Definicijski modul je zaznamovan z besedo **package**, delovni pa z **package body**. Kot v moduli-2 so tudi v adi definirane strukture za oddajo in prevzem tipov, spremenljivk in podprogramov (**use**, **private**). Bistvena razlika med obema jezikom pa je v programskem modulu, ki je pri adi podprogram v moduli-2 pa zopet modul.

V sistemu jezika modula-2 po definiciji obstaja še ena bistvena prednost pred ado. Sistem jezika modula-2 namreč v postopku prevajanja in povezovanja sam preverja, da je modul, ki prevzema neko strukturo iz drugega modula, preveden kasneje. Tega ada v osnovi ne pozna.

3.4 Nizkonivojsko programiranje in vzporednost

Na tem področju je ada v definicijah precej šibkejša od module-2. Adino spremenljivko, na primer, ni možno namestiti na željeno mesto v pomnilniku in programer ima na voljo le še dinamičen način dostopa.

Za programiranje vzporednih postopkov ima ada enoto **task**, ki je podobna procesu v moduli-2. Prednost pred modulo-2 pri vzporednem programiranju pa predstavljajo adine ukazne strukture, ki omogočajo vsklajevanje vzporednih procesov; v moduli-2 mora programer te podprogram napisati sam. Kot modula-2 tudi ada omogoča posluževanje prekinitev.

Collins [14] pravi, da bi bila primerjava module-2 in ade nekaj podobnega kot primerjava med pascalom in PL/1: kar je prepuščeno v moduli-2 programerju, je v adi že vgrajeno v jezik. Tu lahko tudi iščemo vzrok obsežnosti ade.

4. C IN MODULA-2

Modula-2 in jezik c sta zelo različna jezika. Zato se bomo omejili le na primerjanje možnosti za programiranje na spodnjem nivoju, torej na nivoju sistemskega programiranja. Razen tega je smiselna primerjava med obema jezikoma v pogledu prenosljivosti programov, medtem ko bi bila primerjava višjih programskih struktur brezpredmetna, ker jih v c-ju ni. Zato v c-ju ne poznamo principov modularnega programiranja in ločenega prevajanja. Tudi v podatkovnih strukturah je c dokaj reven.

Jezik c je nastal hkrati z operacijskim sistemom Unix [7, 23] ter postal osnovni jezik tega močno razširjenega operacijskega sistema. Zato program napisan v c-ju lahko dandanes brez večjih sprememb poženemo na poljubnem drugem računalniku z operacijskim sistemom Unix. Prav prenosljivosti programov je tisti pojem, po katerem je smiselno primerjati modulo-2 in c. Kot pri jeziku c, je torej tudi pri moduli-2 zagotovljena prenosljivost programov, vendar v primeru programov v moduli-2 ne samo med različnimi računalniki z istim operacijskim sistemom, ampak celo med računalniki z različnimi operacijskimi sistemi. Doseči ta cilj v kar se da veliki meri je naloga delovne skupine projekta OSSI [2]. Točneje, izdelati želijo nekakšen vmesnik med modulo-2 in poljubnim operacijskim sistemom.

Jeziku c običajno pripisujejo moč in prednost predvsem pri nizkonivojskem programiranju, vendar zlahka uvidimo, da ima vse te zmožnosti tudi modula-2, ki omogoča celo večji nabor sistemskih operacij.

Največja moč jezika c je v naslavljanju pomnilnika, vendar prav c ne pozna statične definicije spremenljivke na določenem mestu v pomnilniku. Pač pa c lahko poljubno spremenljivko pretvori v kazalec na naslov (to dosežemo z uporabo posebnega znaka *), kar v moduli-2 dosežemo s klicem funkcije ADDRESS. V c-ju lahko vedno ugotovimo naslov v pomnilniku, kjer se nahaja določena spremenljivka, v moduli-2 pa to opravimo s funkcijo ADR.

Ker c izhaja po svojih definicijah iz zbirnika, ki mu poskuša ostati čimbolj podoben, je temu primerno bogat tudi po načinih naslavljanja. Avtomatično dodajanje (x++) ali odvzemanje (x-=i) v c-ju, je v moduli izvedljivo z dodatnim klicem podprograma INC.

V c-ju obstaja poseben način definiranja posameznih spremenljivk, ki prevajalniku pove, naj poskuša shraniti spremenljivke v procesorske registre (**register**). V moduli-2 to ni potrebno, ker so registri neposredno dostopni, drugi cilj omenjene c-jeve strukture - optimizacija kode, pa je v bistvu problem prevajalnika in ne jezika samega.

Delo s posameznimi biti, ki je v c-ju zelo enostavno, je v moduli-2 izvedljivo s spremenljivkami tipa BITSET in klicem podprogramov INCL ter EXCL.

Za razliko od module-2 pa v c-ju ne moremo uporabljati procesov in tudi posluževanje prekinitev po definiciji ni del jezika [1, 7, 19].

Nizkonivojsko in sistemsko programiranje, za kar je c dokaj primeren, je enako dobro podprto tudi v moduli-2. Ob tem pa nam modula-2 ponuja še vse prednosti strukturiranega programiranja, ločenega prevajanja, preverjanja sovpadanja tipov in še cele vrste prednosti, ki jih zahteva sodoben, učinkovit programski jezik, s čimer se c ne more ravno pohvaliti.

5. Nekateri prevajalniki za modulo-2

V tem poglavju si bomo ogledali tri prevajalnike in en interpreter ter z njimi bomo prehodili kratko zgodovino module-2.

4.1 Prevajalnik M2M in M2RT11

Prevajalnik M2RT11 [10, 11, 12, 15] iz ETH Zurich je bil prvi široko uporabljani prevajalnik za modulo-2; pisan je v moduli-2 za računalnik PDP-11 (LSI-11) pod operacijskim sistemom RT-11. Avtor module-2 Wirth [25] je novi jezik dokončno oblikoval in realiziral skozi ta prevajalnik.

Dokumentacija obsega pet zvezkov, ki opisujejo samo sistem jezika brez primerjave s kakšnim drugim jezikom. V njih tudi zasledimo, da obstaja različica tega prevajalnika za operacijski sistem Unix.

Osnovni prevajalnik (M2RT11) je petprehodovni, kakršni so praviloma tudi vsi drugi. Prav počasnost prevajanja, ki izvira iz števila prehodov, je ena največjih slabosti module-2. Ker so vsi novejši prevajalniki nastali prav na podlagi tega osnovnega prevajalnika, si ga oglejmo podrobneje.

Sistem jezika sloni na kosu programa, ki je napisan v zbirniku in se imenuje Run Time System. V njem so vsebovani vsi podprogrami, ki jih oddaja modul SYSTEM in še nekaj splošnih podprogramov. Klic vseh teh podprogramov je izveden preko programskih prekinitev - pasti. Vedno obstaja možnost preprogramiranja tega dela programa tako, da lahko potem poganjamo program brez podlage operacijskega sistema (stand alone).

V sistem je vključen prevajalnik, ki obsega naslednjih pet prehodov:

1. preverjanje sintaktične pravilnost prevajane enote in branje simbolne datoteke vnešenih modulov
2. preverjanje pravilnosti deklaracij in izdelava referenčne datoteke

3.: preverjanje sovpadanja tipov v telesih podprogramov
4.: generiranje kode za izraze
5.: generiranje kode za posamezne stavke in izdelava izhodne datoteke

Kasneje so definirali neko obliko vmesne kode, podobno kot p-kodo pri pascalu, in jo imenovali M-koda. Prvič je bila ta oblika kode uporabljena pri prevajalniku M2M, kjer sta 4. in 5. prehod združena v enega samega in sicer:

4.: generiranje vmesne, M kode

V dokumentaciji je dodan tudi interpreter te kode in njen opis [13] Sicer pa se vsi prehodi izvedejo le pri prevajanju delovnih in programskih modulov, medtem ko so za definicijske module dovolj že prvi trije.

Poleg prevajalnika vsebuje sistem jezika še povezovalnik in iskalnik napak (debugger) ter vrsto uporabniških modulov z orodji, ki omogočajo delo z datotekami in podobno.

Sistem razpolaga z več vrstami datotek, ki jih loči po podaljških:

- .MOD: programski ali delovni modul
- .DEF: definicijski modul
- .LST: izpis prevedenega programa (potrebuje ga iskalec napak)
- .REF: referenčna tabela (potrebuje jo iskalec napak)
- .LNK: izhodna datoteka iz prevajalnika, ki je primerna za povezovanje
- .LOD: izhodna datoteka iz povezovalnika, ki je primerna za izvajanje v okviru sistema
- M2M in M2S: sistemske datoteke

Vsi naslednji primeri prevajalnikov so nekakšne izpeljanke iz osnovnega M2M.

5.2 Modula-2 za Z80 pod CP/M

Tudi ta prevajalnik prihaja iz Švice, je pa precej okrnjena različica, saj ne pozna pojma procesa in vzporednosti delovanja. Razen tega tudi ne omogoča posluževanja prekinitev. Kot nadomestilo pa ohranja povezavo z zbirnim jezikom: uporabnik lahko po želji določene delovne module napiše v zbirniku.

Sistem sestoji iz podobnih delov kot M2M in zopet je priložena množica delovnih modulov, ki omogočajo delo s samim operacijskim sistemom. Dokumentacija te implementacije sistema modula-2 [14] podrobno opisuje razlike med pascalom in modulo-2.

5.3 Modula-2 razvojni sistem

Je interpreter za modulo-2 za računalnik v tem primeru IBM-PC. Med predstavljenimi primeri je to tudi edini interpreter. Nastal je leta 1984 pri tvrdki Modula Corporation. Ta sistem je pravzaprav celoten prevajalnik, ki pa mu manjka peti prehod, torej generacija kode, ki bi bila izvedljiva na ciljnem računalniku. Implementacija module-2 je dokaj celovita, saj iz definicije jezika manjka samo možnost posluževanja prekinitev (podprogram SYSTEM.IOTRANSFER). Ta sistem je še najbolj primeren za nadaljne razvojno delo na moduli-2, predvsem v smeri razvoja in izdelave petega prehoda, ki bi izdeloval resnično izvedljivo kodo. Možno pa bi bilo tudi dodati še en prehod za optimizacijo M-kode, ki bi ga vstavili med četrti in peti prehod.

5.4 SDS-XP

Sistem SDS-XP sodi trenutno med najboljše prevajalnike za modulo-2 za računalnik IBM-PC in operacijski sistem PC-DOS.

Implementacija module-2 je povsem popolna in vključuje možnost nadzora nad prekinitvami ter vzporedno poganjanje procesov. Pa tudi sicer lahko o sistemu zapišemo samo najboljše, saj poleg običajnih delov vsebuje še poseben urejevalnik, v katerem so definirane posebne tipke, ki omogočajo enostavno pisanje ukaznih struktur (zato tudi odpade prvi prehod) ter možnost uporabe knjižnice modulov, česar ne omogoča nobena druga od opisanih implementacij module-2. Razen tega SDS-XP odlikuje z izredno bogatimi orodji, ki so vključena v sistemski knjižnici ITCLIB.

S tem smo zaključili pregled nekaterih vidnih primerov prevajalnikov za modulo-2. Ta pregled pa ni popoln, saj nismo omenili sistemov za večje računalnike (VAX pod VMS) in podobno.

6. Modula-2 in prolog

Zadnjih nekaj vrstic tega pregleda namenimo zelo zanimivemu paketu za povezavo module-2 in prologa. Razvit je bil v Zurichu pod vodstvom Carla Mullerja [17, 18]. Povezava med obema programskima jezikoma je dvosmerna, saj lahko uporabljamo v prologu podprograme napisane v moduli-2, ali pa prologova načela logičnega iskanja po podatkovni bazi v moduli-2. Prva možnost je zelo enostavna. Definirati moramo le obliko podprograma vsebovanega v nekem modulu. Ta podprogram nam v postopku interpretacije prologovega programa predstavlja nek prologov stavek (klavzulo). Tako lahko resnično pričnemo razvijati nek program v prologu in postopoma stavek za stavkom prevajamo v modulo-2. Na koncu imamo celoten program v moduli-2, ga prevedemo in s tem pridobimo na hitrosti, ki je najbolj šibka točka prologovih interpreterjev.

Obraten, manj pogost primer uporabe, sloni na klicu prologovega interpreterja, ki je samo nek podprogram. Temu podprogramu predamo kot parameter nek niz, ki je v bistvu prologov stavek, katerega interpreter obdela na enak način, kot vsak drug prologov stavek. Opisani sistem je bil razvit na računalniku Lilith, sedaj pa je na volju tudi za računalnike IBM-PC pod MS-DOS in VAX pod VMS.

Enostavna prenosljivost paketa je prav plod uporabe OSSI standarda, ki ga paket uporablja za vse posege proti operacijskemu sistemu.

7. Lastne izkušnje

V Odseku za računalništvo in informatiko Inštituta Jožef Stefan že nekaj časa sistematično zbiramo dokumentacijo in orodja za programiranje v moduli-2. Do sedaj smo v moduli-2 realizirali nekaj manjših poskusnih projektov, od katerih jedra dveh navajamo v dodatku. Med drugim smo v moduli-2 razvili razporejevalnik sistemskega časa in s tem omogočili pisanje resničnih sočasno tekočih procesov na enem procesorju v nasprotju z definicijo module-2, ki pozna samo sorutine.

Naslednji korak pa je implementacija večih sočasno tekočih procesov na večih procesorjih. Projekt razvijamo v okviru razvoja poskusnega okolja PS-11 za programiranje vzporedno delujočih procesorjev na vodilu Q [3, 8]. Dosedanji rezultati so dokaj vzpodbudni: doslej

smo uspeli pognati več vzporednih, vendar zaenkrat nesodelujočih procesov, ki si delijo skupni pomnilnik.

Poleg tega nastaja v naših laboratorijah prva implementacija OSSI standarda za operacijski sistem RT-11 in MicroVMS.

7. Zaključek

Modula-2 se je v preteklih nekaj letih uveljavila kot jezik za sistemsko programiranje in kot jezik za pisanje velikih programskih sistemov. Obstaja ocena, da je v moduli-2 napisane že okoli 15% novejše sistemske programske opreme. Razlog za hitro razširjenje jezika je brez dvoma v njegovi enostavnosti in učinkovitosti. Prav gotovo si bo modula-2 s svojo preglednostjo in majhnostjo samega sistema, ter enostavnostjo rokovanja pridobila v prihodnje še veliko večjo veljavo in že morda kar kmalu nadomestila "stari dobri pascal".

Zahvala

Za vso pomoč pri iskanju novih znanj in dragocene nasvete pri oblikovanju besedila se najlepše zahvaljujem prof.dr.Boštjanu Vilfanu.

Literatura

[1] V.Batagelj, A.Brodnik in ostali: Proceedings of 4th Summer School of Computer Sciences, IJS Report 4352, Ljubljana, 1986.
[2] E.Biagioni, K.Hinrich, G.Heiser, C.Muller: A Portable Operating System interface and Utility Library, IEEE Software, November 1986.
[3] A.Brodnik, S.Mavrič, R.Trobec: Q-bus Base Multiprocessor System, CompEuro, Hamburg, 1987.
[4] S.Collins, B.Marshall, N.King: Modula-2 - A Babel Fish for Chips?, System International, December 1986.
[5] R.Gleaves: Modula-2 for Pascal Programmers, Springer-Verlag, 1984.
[6] K.Jensen, N.Wirth: PASCAL User Manual and Report, Springer-Verlag, 1975.
[7] B.W.Kernighan, D.M.Ritchie: The C Programming Language, Prentice-Hall, 1978.
[8] S.Mavrič, A.Brodnik, R.Trobec, M.Spegel, P.Kolbezen: PS-11: Večprocesorski sistem na vodilu Q, Mipro 1987, Opatija, 1987.
[9] Modula-2 Development System, Modula Corporation Provo, Utah, 1984.
[10] Modula-2 Installation Instructions, ETH Institut fur Informatik, Zurich, 1981.
[11] Modula-2 Overview of the Modula-2 Compiler, ETH Institut fur Informatik, Zurich, 1981.
[12] Modula-2 Overview of Modula-2 Debugger, ETH Institut fur Informatik, Zurich, 1981.
[13] Modula-2 Overview of Modula-2 System M2M, ETH Institut fur Informatik, Zurich, 1982.
[14] Modula-2 System for Z80 CP/M, Hochstrasser Computing AG, Zurich, Switzerland, 1984.
[15] Modula-2 User Guide, ETH Institut fur Informatik, Zurich, 1981.
[16] B.Mohar, E.Zakrajšek: Uvod v programiranje, DMFA, Ljubljana, 1982.
[17] C.Muller: Modula-Prolog (User Manual), ETH Institut fur Informatik, Zurich, Juli 1985.
[18] C.Muller: Modula--Prolog: A Software Development Tool, IEEE Software, November 1986.
[19] Osnove programskega jezika C, Delovni zvezek Izobraževalni center Delta, 1984.
[20] G.Pomberger: Software Engineering and Modula-2, Prentice-Hall, 1984.
[21] I.C.Pyle: The Ada Programming Language, Prentice-Hall, 1981.
[22] SDS-XP, Extended Performance Modula-2 Software Development System, Interface Technologies, Houston, Texas, 1985.
[23] R.Thomas, L.R.Rogers, J.L.Yates: Advanced Programmer's Guide to UNIX System V, McGraw-Hill, 1986.
[24] P.Wegner: Programming with Ada, Prentice-Hall, 1980.
[25] N.Wirth: Programming in Modula-2, Third, Corrected Edition, Springer-Verlag, 1985.

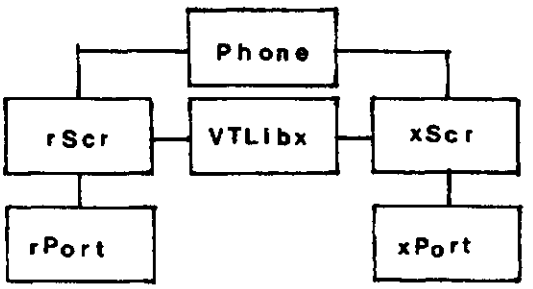
Dodatek A - Phone

Program Phone omogoča pogovor med dvema terminaloma priključenima na računalnik LSI-11 pod operacijskim sistemom RT-11. Ker je RT-11 enouporabniški sistem, je lahko hkrati aktiven le en terminal. Tu program poženemo in izberemo drugega. Pri obeh se na zaslonu pojavi obrazec. Na zgornjo polovico zaslona se izpisujejo znaki, sprejeti od drugega terminala, na spodnjo pa znaki, ki jih terminal pošilja drugemu.

Program sestoji iz šestih modulov:

- **Phone:** osnovni program
- **rScr:** modul za delo z enim zaslonom
- **xScr:** skrbi za drugi zaslon
- **VTLibx:** posebni ukazi za delo s terminali tipa VT100
- **rPort:** posluževanje prekinitev na enem vhodu
- **xPort:** posluževanje prekinitev na drugem vhodu

Na sliki 1 je še prikazana podrobnejša shema medsebojne odvisnosti posameznih modulov.



slika 1

Definicijski moduli:

```
DEFINITION MODULE rScr;
  (* modul za delo s prvim terminalom *)
  EXPORT QUALIFIED
    rPutCh, rPripraviEkran, rBrisi, rGetCh;

  PROCEDURE rPutCh (ch: CHAR; flag: BOOLEAN);
    (* izpiše znak na prvi zaslon *)
  PROCEDURE rPripraviEkran (VAR rVect,
                            rCSR: CARDINAL);
    (* pripravi zaslon terminala *)
  PROCEDURE rBrisi;
    (* zbriše zaslon terminala *)
  PROCEDURE rGetCh (VAR ch: CHAR;
                    VAR OK: BOOLEAN);
    (* pobere zlog s terminala *)
END rScr.

DEFINITION MODULE VTLibx;
  (* Ukazi za delo s terminali tipa VT 100 *)
  EXPORT QUALIFIED sCursor, sErase, sWriteLine,
                   sScroll, sNewLine, entire,
                   PutString, Obrazec, DelCh;

  CONST   entire = 2; (* zbriše ves zaslon *)
```

```
  TYPE    WriteProc = PROCEDURE (CHAR);
    (* podprogram za izpis zloga *)

  PROCEDURE sScroll (from, to: CARDINAL;
                     write: WriteProc);
    (* določi območje premikanja zaslona *)
  PROCEDURE sErase (area: CARDINAL;
                    write: WriteProc);
    (* zbriše določen del zaslona *)
  PROCEDURE sCursor (x, y: CARDINAL;
                     write: WriteProc);
    (* postavi slednik na določeno mesto *)
  PROCEDURE sNewLine (write: WriteProc);
    (* premakne slednik v novo vrsto *)
  PROCEDURE sWriteLine (long: CARDINAL;
                        write: WriteProc);
    (* izpiše črto zahtevane dolžine *)
  PROCEDURE PutString (what: ARRAY OF CHAR;
                       write: WriteProc);
    (* izpiše niz *)
  PROCEDURE Obrazec (write: WriteProc);
    (* izpiše osnovni zaslonski obrazec *)
  PROCEDURE DelCh (write: WriteProc);
    (* zbriše zadnji znak v tekoči vrstici *)
END VTLibx.

DEFINITION MODULE xPort;
  (* posluževanje prekinitev na kanalu *)
  EXPORT QUALIFIED
    xGetByt, xInit, xClose, xPutByt;

  PROCEDURE xGetByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* pobere zlog iz vmesnika,
       če je na voljo *)
  PROCEDURE xInit (VAR xVect, xCSR: CARDINAL);
    (* nastavi vse prekinitvene podprograme *)
  PROCEDURE xClose;
    (* zapre in vse pospravi okoli kanala *)
  PROCEDURE xPutByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* vstavi znak v vmesnik, če je prostor *)
END xPort.
```

V primeru modulov rScr in xPort smo navedli samo po en modul za vsako stran. Druga sta namreč skoraj povsem enaka. Sledi še podoben pregled programskega in delovnih modulov. Navajamo samo delovna modula, ki sta zanimivejša (izpuščamo VTLibx).

```
MODULE Phone;
  (* glavni programski modul *)
  FROM xScr IMPORT
    xGetCh, xPutCh, xPripraviEkran, xBrisi;
  FROM rScr IMPORT
    rGetCh, rPutCh, rPripraviEkran, rBrisi;
  FROM TTIO IMPORT
    WriteString, Read, Write, SetMode;

  CONST ETX = 003C;   (* <CTRL><C> *)
  VAR xch, rch: CHAR;
      OK, OK: BOOLEAN;
      CSR, Vect: CARDINAL;

  PROCEDURE BeriCSR (niz: ARRAY OF CHAR;
                     VAR Vect, CSR: CARDINAL);
  (* določi vektorja in naslove statusnih
     registrov obeh terminalov *)
  VAR Enota: CHAR;
  BEGIN
    WriteString (niz);
    Read (Enota); Write (Enota);
    CASE Enota OF
        '1': Vect := 310B; CSR := 1765108
      | '2': Vect := 350B; CSR := 1765508
    END
  END BeriCSR;

BEGIN
  SetMode (0, FALSE);  (* prepove <CTRL><C> *)
  BeriCSR ('    Od TT (st.): ', Vect, CSR);
  xPripraviEkran (Vect, CSR);
  BeriCSR ('    Proti TT (st.): ', Vect, CSR);
  rPripraviEkran (Vect, CSR);
  REPEAT (* glavna zanka za izmenjavo znakov *)
    xGetCh (xch, OK);
    IF OK THEN
      rPutCh (xch, TRUE); xPutCh (xch, FALSE)
    END;
    rGetCh (rch, OK);
    IF OK THEN
      xPutCh (rch, TRUE); rPutCh (rch, FALSE)
    END
  UNTIL (xch = ETX) OR (rch = ETX);
  xBrisi; rBrisi
END Phone.

IMPLEMENTATION MODULE rScr;
  (* delo s sprejemnim zaslonom *)
  FROM rPort IMPORT
    rInit, rGetByt, rClose, rPutByt;
  FROM VTLibx IMPORT
    sCursor, sErase, sWriteLine, sScroll,
    sNewLine, entire, PutString, DelCh,
    Obrazec;

  CONST CR = 015C; LF = 012C; DEL = 177C;
        BS = 010C; REFR = 027C; ESC = 033C;

  VAR xm, xt, ym, yt, x, y: CARDINAL;

  PROCEDURE Send (ch: CHAR);
    (* pošlje znak na zaslon *)
  VAR ok: BOOLEAN;
  BEGIN
    REPEAT rPutByt (ch, ok) UNTIL ok
  END Send;

  PROCEDURE rPripraviEkran (VAR rVect,
                                rCSR: CARDINAL);
    (* nastavi terminal in izpiše obrazec *)
  BEGIN
    rInit (rVect, rCSR);
    xm := 1; xt := 1; ym := 11; yt := 22;
    x := xm; y := ym;
    Obrazec (Send)
  END rPripraviEkran;

  PROCEDURE rPutCh (ch: CHAR; flag: BOOLEAN);
    (* interpretira sprejeti znak *)
    PROCEDURE Uredi (VAR a, b: CARDINAL);
    VAR i: CARDINAL;
    BEGIN
      IF ch = REFR THEN
        FOR i:=1 TO 10 DO sNewLine (Send) END;
        a := 1
      END;
      IF ch = DEL THEN DelCh (Send); a:=a-1
      ELSIF ch = CR THEN sNewLine (Send); a:=1
      ELSE Send (ch); a := a + 1 END;
    END Uredi;
  BEGIN
    IF (flag) THEN
      IF x # xm THEN
        sScroll (3, 11, Send);
        sCursor (xm, ym, Send);
        x := xm
      END;
      Uredi (xm, ym)
    ELSE
      IF x # xt THEN
        sScroll (14, 22, Send);
        sCursor (xt, yt, Send);
        x := xt
      END;
      Uredi (xt, yt)
    END;
  END rPutCh;

  PROCEDURE rGetCh (VAR ch: CHAR;
                    VAR OK: BOOLEAN);
    (* dobi znak s terminala *)
  BEGIN
    rGetByt (ch, OK);
  END rGetCh;

  PROCEDURE rBrisi;
    (* zbriše ekran in zaključi komunikacijo *)
```

```
  BEGIN
    sErase (entire, Send); rClose
  END rBrisi;
END rScr.


IMPLEMENTATION MODULE xPort [7];
  (* podprogrami za delo z enim od kanalov *)

  FROM SYSTEM IMPORT
    ADR, ADDRESS, SIZE, WORD,
    NEWPROCESS, TRANSFER, IOTRANSFER;

  CONST N = 32;

  TYPE DLV11J  = RECORD
                   rCSR: BITSET;
                   rBUF: CARDINAL;
                   xCSR: BITSET;
                   xBUF: CARDINAL
                 END;

       TypeCSR = POINTER TO DLV11J;

  VAR rn, rin, rout, xn, xin, xout: CARDINAL;
      rbuf, xbuf: ARRAY [0..N-1] OF CHAR;
      rPRO, xPRO, CON: ADDRESS;
      rWSp, xWSp: ARRAY [0..177B] OF WORD;

      px1, px2, px3, px4: WORD;

      active: BOOLEAN;
      Vector: ADDRESS;
      CSR: TypeCSR;

  PROCEDURE xGetByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* pobere zlog iz sprejemnega vmesnika,
       če lahko *)
  BEGIN
    IF rn > 0 THEN
      OK := TRUE;
      ch := rbuf [rout]; rn := rn - 1;
      rout := (rout+1) MOD N;
    ELSE OK := FALSE
    END
  END xGetByt;

  PROCEDURE producer;
    (* sprejemni prekinitveni podprogram *)
  BEGIN
    LOOP
      IOTRANSFER (rPRO, CON, Vector);
      IF rn < N THEN
        rbuf [rin] := CHR (CSR^.rBUF);
        rin := (rin+1) MOD N;
        rn := rn + 1
      END
    END
  END producer;

  PROCEDURE xPutByt (VAR ch: CHAR;
                     VAR OK: BOOLEAN);
    (* vstavi zlog v oddajni vmesnik,
       če lahko *)
  BEGIN
    IF xn < N THEN
      OK := TRUE;
      xbuf [xin] := ch; xn := xn + 1;
      xin := (xin+1) MOD N;
      IF NOT active THEN
        INCL (CSR^.xCSR, 6); active := TRUE
      END
    ELSE OK := FALSE;
    END
  END xPutByt;

  PROCEDURE consumer;
    (* oddajni prekinitveni podprogram *)
  BEGIN
    LOOP
      IOTRANSFER (xPRO, CON,
                  ADDRESS (Vector+4));
      IF xn > 0 THEN
        CSR^.xBUF := ORD (xbuf [xout]);
        xn := xn - 1; xout := (xout+1) MOD N
      ELSE
        EXCL (CSR^.xCSR, 6); active := FALSE
      END
    END
  END consumer;

  PROCEDURE xInit (VAR xVect, xCSR: CARDINAL);
    (* inicializiramo registre, ki jih bomo
       potrebovali in poženemo nov proces *)
  VAR a: ADDRESS;
  BEGIN
    Vector := xVect; CSR := TypeCSR (xCSR);

      (* shranimo stare vrednosti *)
    a := ADDRESS (Vector);
    px1 := a^; INC (a, 2);
    px2 := a^; INC (a, 2);
    px3 := a^; INC (a, 2);
    px4 := a^;

    rn := 0; rin := 0; rout := 0;
    NEWPROCESS (producer, ADR(rWSp),
                SIZE(rWSp), rPRO);

    xn:=0; xin:=0; xout:=0; active:=FALSE;
    NEWPROCESS (consumer, ADR(xWSp),
                SIZE(xWSp), xPRO);

    WITH CSR^ DO
      INCL (rCSR, 6); TRANSFER (CON, rPRO);
      TRANSFER (CON, xPRO)
    END
  END xInit;

  PROCEDURE xClose;
    (* počistimo vse za seboj *)
  VAR a: ADDRESS;
  BEGIN
      (* obnovimo stare vrednosti *)
    a := ADDRESS (Vector);
    a^ := px1; INC (a, 2);
    a^ := px2; INC (a, 2);
    a^ := px3; INC (a, 2);
    a^ := px4
  END xClose;
END xPort.
```

Na sliki 2 pa podajamo še prikaz medsebojnih klicev modulov. Primer je tudi lep način razbitja na module in definicije njihovih stičnih točk - podprogramov.

Dodatek B - Razporejevalnik

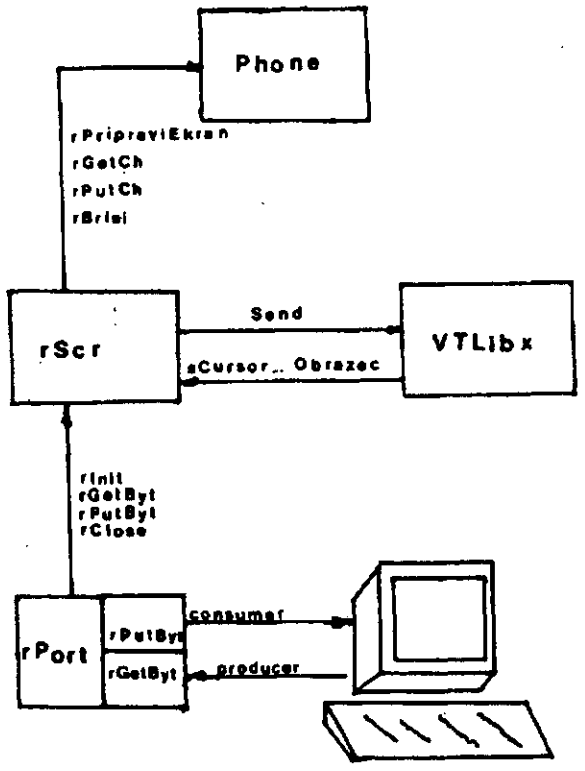
Tukaj navajamo samo primer definicijskega in delovnega modula razporejevlanika procesorskega časa in kratek primer z dvema procesoma.

```
DEFINITION MODULE Scheduler;
  EXPORT QUALIFIED ProcessGo;
  PROCEDURE ProcessGo (which: PROC;
                       space: CARDINAL);
    (* pozene nov proces, ki uporablja space
       velik prostor v pomnilniku *)
END Scheduler.

IMPLEMENTATION MODULE Scheduler[7];

  FROM SYSTEM IMPORT
    ADDRESS, PROCESS,
    NEWPROCESS, TRANSFER, IOTRANSFER;
  FROM Storage IMPORT ALLOCATE;

  CONST N = 10;
        (* najvecje stevilo procesov *)
        ClockVec = 100B;
          (* prekinitveni vektor ure *)
        TimerSpace = 200B;
          (* prostor za prekinitveni
```



slika 2

```
          podprogram za uro *)
      TicksPerPeriod = 50
        (* stevilo urinih enot, ki so
           dodeljeni enemu procesu *)

VAR StartNewProcess: BOOLEAN;
    wsp: ADDRESS;
    FirstProc,          (* starting process *)
    TimerAdd: PROCESS;(* here is timer *)
    Processes: ARRAY [1..N] OF PROCESS;
    n, current: CARDINAL;

PROCEDURE Timer;
  (* dusa razporejevlnika *)
VAR NumberOfTicks: CARDINAL;
BEGIN
  NumberOfTicks := TicksPerPeriod;
  LOOP
    IF n > 0 THEN
      IF StartNewProcess THEN
        StartNewProcess := FALSE;
        NumberOfTicks := TicksPerPeriod;
        current:=n
      ELSIF NumberOfTicks <= 0 THEN
        NumberOfTicks := TicksPerPeriod;
        current:=(current MOD n)+1
      ELSE
        NumberOfTicks := NumberOfTicks - 1
      END;
      IOTRANSFER (TimerAdd,
                  Processes[current],
                  ClockVec)
    ELSE
      NumberOfTicks := TicksPerPeriod;
      IOTRANSFER (TimerAdd, FirstProc,
                  ClockVec)
    END;
  END
END Timer;

PROCEDURE ProcessGo (which: PROC;
                     space: CARDINAL);
  (* pozene nov proces *)
VAR wsp: ADDRESS;
    OldCur: CARDINAL;
BEGIN
  IF n < N THEN
    ALLOCATE (wsp, space);
      (* poiscemo prostor za novi proces *)
    n:=n+1;
    NEWPROCESS (which, wsp, space,
                Processes[n]);
    StartNewProcess := TRUE;
    IF n = 1 THEN          (* ali je prvi *)
      TRANSFER (FirstProc, TimerAdd)
    ELSE          (* sicer ga pozene Timer *)
      TRANSFER (Processes[current],
                TimerAdd)
    END;
  END
END ProcessGo;

BEGIN
  n:=0; current:=1; StartNewProcess := FALSE;
  ALLOCATE (wsp, TimerSpace);
  NEWPROCESS (Timer, wsp, TimerSpace,
              TimerAdd);
  TRANSFER (FirstProc, TimerAdd)
    (* in pozeneo Timer *)
END Scheduler.

MODULE TestScheduler;

  FROM Scheduler IMPORT ProcessGo;
  FROM VTLibx IMPORT sCursor;
  FROM TTIO IMPORT Write, Read;

  CONST SpaceNeeded = 4008;
  VAR ch: CHAR;

  PROCEDURE One;
    (* na zaslon izpisuje "A" *)
  VAR i: CARDINAL;
  BEGIN
    ProcessGo (Two, SpaceNeeded); i:=1;
    LOOP
      sCursor (i, 4, Write); i:=(i MOD 80)+1;
      Write ('A')
    END
  END One;

  PROCEDURE Two;
    (* ta pa naj izpisuje "B" *)
  VAR i: CARDINAL;
  BEGIN
    i:=1;
    LOOP
      sCursor (i, 5, Write); i:=(i MOD 80)+1;
      Write ('B')
    END
  END Two;

BEGIN
  ProcessGo (One, SpaceNeeded)
END TestScheduler.
```