STICHTING
MATHEMATISCH CENTRUM
2e BOERHAAVESTRAAT 49
AMSTERDAM
REKENAFDELING

Sequential access files voor de EL X8

deel 1:

Voorstel voor de automatic routines

door

J.V.M. van der Grinten
P.J.W. ten Hagen
F.E.J. Kruseman Aretz

NR 7

februari 1969

The Mathematical Centre at Amsterdam, founded the 11th of February, 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications, and is sponsored by the Netherlands Government through the Netherlands Organization for Pure Research (Z.W.O.) and the Central National Council for Applied Scientific Research in the Netherlands (T.N.O.), by the Municipality of Amsterdam and by several industries.

VOORWOORD

Als gemeenschappelijk project van het ERC te Utrecht en het MC te Amsterdam wordt een multiple-access bedrijfssysteem ontwikkeld voor de EL X8-installatie, uitgebreid met een PDP-8/I als communicatie satelliet.

Aan dit bedrijfssysteem ligt een file-systeem ten grondslag: alle in het systeem aanwezige informatie is hierbij georganiseerd in files van samenhangende brokken. Deze zullen i.h.a. bewaard worden in een of andere vorm van achtergrondsgeheugen (trommel, schijven, magneetbanden).

Voor de (ALGOL 60-) gebruiker staat het file-systeem ter beschikking voor input- en output en als werkruimte, zonder dat hij weet hoeft te hebben van of invloed kan uitoefenen op de wijze waarop het bedrijfssysteem zijn files beheert.

Deze notitie behelst de definitie van sequential access-files (d.w.z. die file-soorten waarvan de informatie slechts in een zekere volgorde toegankelijk is), zoals zij zich aan de gebruiker manifesteren.

In het bijzonder worden daartoe de atomaire gebruikersprocedures, voor een sequential access file geformuleerd op een machine- en medium-onafhankelijke wijze.

De conclusies, die deze notitie beschrijft zijn het resultaat van uitgebreide discussies in de groep die werkt aan bovengenoemd project. Aan de vergadering van deze werkgroep wordt actief deelgenomen door vertegenwoordigers van Philips Nat. Lab. Waalre en het Psych. Research Lab. van de VU.

Definitieve voorstel sequential access files

1. Inleiding:

Een sequential access file (s.a. file) is een stuk geheugen dat door de gebruiker sequentieel bespeeld kan worden. Ieder contact tussen gebruiker (programma) en s.a.file loopt via het kerngeheugen van de X8. We kunnen ons daarom de s.a.file voorstellen als een lineaire string van X8-woorden, waaruit zowel van voor naar achter als van achter naar voor gelezen kan worden.
Het schrijven in een s.a.file gebeurt steeds door de nieuwe informatie achter aan de bestaande string te hechten.
In dit voorstel wordt niet ingegaan op de principieel eindige lengte van een file.
In de volgende definities van s.a.file-manipulaties wordt op geen enkele wijze kennis vooronderstelt van de manier waarop de files in de kernen of elders in het achtergrondgeheugen gerepresenteerd zijn.

2. Soorten s.a. files:

Een file bestaat uit een string van informatie-eenheden (symbolen).
Een file-soort wordt bepaald door de wijze waarop de symbolen in één X8-woord (27 bits) zijn opgeborgen.
De file-soorten zijn:

soort no. 1 = sym-soort: 8 bits per symbool, 3 symbolen / woord.
In een sym-file zijn de symbolen gecodeerd volgens de standaard (ALGOL) interne representatie.

soort no. 2 = ½-soort: 54 bits per informatie-eenheid, één informatie-eenheid per 2 X8 woorden.
(b.v. floating point getallen.)

soort no. 3 = 1-soort: 27 bits per informatie-eenheid,
1 inf. eenh./woord (b.v. integers).

soort no. 4 = 2-soort: 12 bits per informatie-eenheid,
2 inf. eenh. / woord (b.v. kaartkolommen).

soort no. 5 = 3-soort: 8 bits per informatie-eenheid,
3 inf. eenh. /woord (b.v. 7- en 8-gatscode).

soort no. 6 = 4-soort: 6 bits per informatie-eenheid,
4 inf. eenh. / woord (b.v. 5- en 6-gatscode).

De soorten no.2 t/m 6 heten ook bin soorten, in tegenstelling met soort no. 1 (sym soort).
De soort van een s.a.file is vanuit een ALGOL-programma opvraagbaar, maar kan niet gewijzigd worden, tenzij de file leeg is (zie 3.).

3. Pointers op s.a.files:

Elke s.a.file heeft 3 pointers n.l.:

1. beginpointer (bp). De bp wijst het begin van de file aan.
2. leespointer (rp). De leespointer wijst het eerstvolgende te lezen karakter aan, bij lezen van voor naar achter (= in de richting van begin naar einde).
3. eindpointer (ep). De eindpointer wijst de plaats direct na de laatste informatie-eenheid van de file aan.
   Bij de aanvraag van de executie van een programma is de leespointer voor elke s.a.filë geïnitialiseerd op de beginpointer van die file.

De lengte van de s.a.file is per definitie gelijk aan ep - bp; d.i. het aantal karakters tussen begin- en eindpointer.
Steeds geldt: $bp \leq rp \leq ep$.
Een s.a.file heet leeg als ep = bp.
De schijnbare of effectieve lengte van een s.a.file is ep - rp, dus het aantal karakters tussen lees- en eindpointer.
Beide lengtes zijn vanuit het ALGOL-programma opvraagbaar en per definitie non-negatief.

De drie genoemde pointers worden gedurende de uitvoering van een ALGOL-programma gewijzigd als gevolg van:

1. lezen uit een file van voor naar achter
   (effect: rp := bp := rp + 1;),
2. bewarend lezen uit de file van voor naar achter
   (effect: rp := rp + 1;),
3. schrijven in de file
   (effect: ep := ep + 1;),
4. lezen uit de file van achter naar voor
   (ontstapelend lezen)
   (effect: ep := ep - 1;),

5. de vernietiging van de eerste k symbolen van een file.
   (effect: bp := bp + k;),
6. terugzetten van de leespointer
   (effect: rp := bp;),
7. clearen van een file
   (effect: ep := rp := bp).

In de volgende secties zullen de procedures, die deze acties veroorzaken, gedetailleerd beschreven worden.

4. s.a.file en programma

Ieder programma mag zonder meer 6 s.a.files gebruiken, deze worden genummerd van 1 t/m 6 (of: 1 t/m max numb of s.a.files). De behandeling van de eigenlijke lees- en schrijf-routines wordt gegeven in sectie 5. Verder staan de volgende primitieve procedures ter beschikking:

```
procedure clear file(i) ; value i ; integer i ;
if i > 0 ∧ i ≤ max numb of s.a.files
then begin < van de i-de file wordt de inhoud vernietigd> ;
           rp[i] := ep[i] := bp[i] ;
           comment ep[i] is de eindpointer van de i-de
             file, rp[i], bp[i] analoog;
     end
else errormessage (wrong sa number) ;

integer procedure sa species(i) ; value i ; integer i ;
if i > 0 ∧ i ≤ max numb of sa files
then sa species := <`file soort no. van de i-de file>
else errormessage (wrong sa number);

integer procedure sa length(i) ; value i ; integer i ;
if i > 0 ∧ i ≤ max numb of sa files
then sa length := ep[i] - bp[i]
else errormessage (wrong sa number);
```

```
procedure change sa species(i, k) ; value i, k ; integer i, k ;
if sa length(i)  = 0
then begin if k > 0 ∧ k ≤ 6 then < geef de i-de file het
                                        soort no k >
                                else errormessage (undefined species)
     end
else errormessage (not empty);

procedure reset rp(i); value i ; integer i ;
if i > 0 ∧ i ≤ max numb of sa files
then rp[i]  := bp[i]
else errormessage (wrong sa number);

procedure clear from left(i, k); value i, k ; integer i, k ;
if k > 0
then begin if k ≥ sa length(i) then clear file(i)
else begin bp[i] := bp[i] + k ;
           if rp[i] < bp[i] then  reset rp[i]
     end
end;

integer procedure apparent sa length(i) ; value i ; integer i ;
if i > , ∧ i ≤ max numb of sa files
then apparent sa length := ep[i] - rp[i]
else errormessage (wrong sa number) ;
```

5. De lees- en schrijfroutines

5.1. De leesroutines:

Er zijn 6 algemene leesroutines voor s.a.files, en wel 3 om uit een sym-file te lezen en 3 om uit een bin-file te lezen. Ze hebben ieder één parameter n.l. het nummer van de s.a.file. Men kan 3 gevallen onderscheiden naar de manier waarop de pointers gebruikt en gewijzigd worden:

1. lezen van voor naar achter (dat is: destructief lezen),
2. bewarend lezen van voor naar achter,
3. lezen van achter naar voor (dat is:ontstapelend lezen).

Bij iedere leesopdracht wordt eerst gecontroleerd of de file-soort (sym of bin) overeenstemt met de leesopdracht (sym-lezen of bin-lezen), zo niet dan volgt een foutmelding.
Aan de procedure identifier kan de gebruiker zien of het sym dan wel bin lezen betreft.

```
integer procedure in sym(i) ; value i ; integer i ;
begin in sym :=  inspectsym(i) ; bp[i] := rp[i]
end ;

real procedure in bin(i) ; value i ; integer i ;
begin in bin := inspectbin(i) ; bp[i] := rp[i] end ;

comment alle bin-procedures zijn van het type real omdat
mogelijk uit een ½-soort gelezen wordt;

integer procedure inspectsym(i) ; value i ; integer i ;
if apparent sa length(i) > 0
then begin if sa species(i) = 1
           then begin inspectsym := < karakter aangewezen
                                      door rp[i] > ;
                      rp[i] := rp[i] + 1
                end
           else errormessage (wrong sa species)
     end
else errormessage (file exhausted);

real procedure inspect bin(i) ; value i ; integer i ;
if apparent sa length(i) > 0
then begin if sa species ≠ 1
           then begin inspectbin := < karakter aangewezen
                                      door rp[i] > ;
                      rp[i] := rp[i] + 1
                end
           else errormessage (wrong sa species)
     end
else errormessage (file exhausted);
```

```
integer procedure unstacksym(i) ; value i ; integer i ;
if sa length(i) > 0
then begin if sa species (i) = 1
           then begin ep[i] := ep[i] - 1 ;
                      unstacksym := < karakter aangewezen
                                      door ep[i] > ;
                      if rp[i] > ep[i] then rp[i]:= ep[i]
                end

           else errormessage (wrong sa species)
      end
else errormessage (stack file exhausted);

real procedure unstackbin(i) ; value i ; integer i ;
if sa length(i) > 0
then begin if sa species(i) ≠ 1
           then begin ep[i] := ep[i] - 1 ;
                      unstackbin := < karakter aangewezen
                                      door ep[i] > ;
                      if rp[i] > ep[i] then rp[i] := ep[i]
                end
           else errormessage (wrong sa species)
     end
else errormessage (stack file exhausted) ;
```

Voor filenummers 1 t/m 6 bestaan specifieke leesprocedures zonder parameter. Dan is het file no. opgenomen in de procedure identifier: i staat voor 1, 2, 3, 4, 5 of 6.

```
integer procedure insym i ; insym i := insym(i) ;
real procedure inbin i ; inbin i := inbin(i) ;
integer procedure inspectsym i ; inspectsym i := inspectsym(i) ;
real procedure inspectbin i ; inspectbin i := inspectbin(i) ;
integer procedure unstacksym i ; unstacksym i := unstacksym(i) ;
real procedure unstackbin i ; unstackbin i := unstackbin(i) ;
```

5.2. De Schrijfroutines

Er zijn twee algemene schrijfroutines op een s.a.file, elk met twee parameters, die respectievelijk het filenummer en de weg te schrijven informatie meegeven. Hiernaast bestaan voor de files met filenummer 1 t/m 6 ook specifieke schrijfroutines met slechts één parameter (de weg te schrijven informatie). Het filenummer volgt dan zoals bij de schrijfroutines uit de identifier.
Bij elke aanroep wordt getest, of de filesoort van de file compatible is met de schrijfroutine. Zonodig wordt de meegegeven informatie tot een geheel getal afgerond en zonodig gereduceerd modulo een tweemacht die van de filesoort (d.w.z. van de pakking per woord) afhangt.
Hierbij wordt een resultaat 0 altijd geïnterpreteerd als + 0, behalve bij bin-files van $\frac{1}{2}$-soort en 1-soort, in welk geval het teken van de actuele parameter behouden blijft.

```
procedure outsym(i, kar) ; value i, kar ; integer i, kar ;
if sa species(i) = 1
then begin kar := kar - kar : 512 * 512 ;
           < berg kar in de i-de file op, op de plaats
             aangewezen door ep[i] > ;
           ep[i] := ep[i] + 1
     end
else errormessage (wrong sa species) ;

procedure outbin (i, kar); value i, kar ; integer i ; real kar ;
if sa species(i) ≠ 1
then begin if sa species(i) > 2
           then begin kar := entier (kar + 0.5) ;
                      if sa species(i) = 4
                      then kar := kar - kar : 4096 * 4096
                      else if sa species(i) = 5
                           then kar := kar - kar : 256 * 256
                           else if sa species(i) = 6
                                then kar := kar - kar : 64 * 64 ;
                end ;
           <berg kar in de i-de file op, op de plaats aangewezen
            door ep[i] > ;
           ep[i] := ep[i] + 1
     end
else errormessage (wrong sa species) ;
```

Als i staat voor 1, 2, 3, 4, 5 of 6 hebben we nog:

```
procedure outsym i(kar) ; value kar ; integer kar ;
outsym (i, kar) ;
procedure outbin i(kar) ; value kar ; real kar ;
outbin (i, kar) ;
```

6. Discussie

6.1. In het hier gepresenteerde file-ontwerp is er bewust naar gestreefd één universeel file-begrip in te voeren en niet, zoals gebruikelijk is, onderscheid te maken tussen b.v. input- en output-files. Dat onderscheid komt pas tot uiting in de manier waarop een programma met een file werkt.

a. De klassieke inputfile is dan te beschouwen als een van te voren gevulde file, die door het programma één keer vernietigend wordt gelezen van begin tot einde. (Opm.: rp geïnitialiseerd op bp.)

b. De klassieke outputfile is een aan het begin lege file (ep = rp = bp) waarin gedurende het programma uitsluitend wordt geschreven.

c. De werkfile, is een nieuw concept, dat echter in diverse vormen reeds gesimuleerd wordt met TO DRUM - FROM DRUM - routines. Toepassingen b.v. sort-merge routines.

In het huidige voorstel kan elke file voor alle doeleinden, ook tegelijkertijd, gebruikt worden. Output kan aan een reeds bestaande (niet-lege) file worden toegevoegd en desgewenst ook weer worden teruggelezen. Verder kunnen files ook gebruikt worden als last-in-first-out buffers of als first-in-first-out buffers.

6.2. Teneinde bestaande programma's, voor zover ze

- wat de invoer betreft uitsluitend gebruikmaken van hetzij resym en read, hetzij van rehep
- wat de uitvoer betreft naast alle printer-routines, uitsluitend gebruikmaken van hetzij puhep, hetzij pusym en hogere pons-procedures

ongewijzigd te kunnen gebruiken, zijn in de bibliotheek de volgende declaraties opgenomen:

```
integer procedure resym ; resym := insym 1 ;
integer procedure rehep ; rehep := inbin 1 ;
```

De invoer gegevens zullen dan vooraf ingelezen moeten zijn in een file (van de juiste soort!), die filenummer 1 krijgt.

procedure prsym(k) ; value k ; integer k ; outsym 2(k) ;

Voor de regeldrukker moet een (lege) file ter beschikking staan van de symsoort, die filenummer 2 krijgt.

procedure pusym(k) ; value k ; integer k ; outbin 3(k) ;

De uitvoer voor de ponser wordt geschreven in de van te voren aangehangen file (van de juiste soort!) met filenummer 3.

7. Standaard uitvoeringen

Teneinde de programmeurs, die van het filesysteem slechts het meest elementaire gebruik maken (hieronder vallen alle huidige gebruikers) niet nodeloos te belasten met het schrijven van metaprogramma's, dienen er een aantal standaard metaprogramma's voor batch-processing te bestaan.
Een voorbeeld hiervan is een metaprogramma, dat

1. een programmafile introduceert en daarin de programmatekst opneemt,
2. een data file introduceert van symsoort en daar de data in opneemt,
3. een empty file van symsoort introduceert en bestemt voor de regeldrukker,
4. idem voor de bandponser,
5. drie lege werkfiles van symsoort introduceert,
6. het geheel ter uitvoering aan het operating system aanbiedt, en na beëindiging van het programma de files genoemd onder 3. en 4., mits ze niet leeg zijn, aanbiedt aan resp. regeldrukker en bandponser.

Overzicht s.a. file routines

procedure aanroep	parameters	test op fouten	effect op de pointers	indien type proc. dan welk assignment aan proc. id.
clear file	file nummer	$0 < i \leq$ max numb of files (wrong filenumber?)	rp := ep := bp	-
sa species	"	"	-	soort no van de i-de file
sa length	"	"	-	ep - bp
apparent sa length	file nummer	wrong file number?	-	ep - rp
change sa species	file nummer, nieuw file soort no.	is file leeg? is soort no. acceptabel?	-	-
reset rp	file nummer	wrong file number?	rp := bp	-
clear from left	file nummer, aantal te vernietigen karakters	wrong file number? (n.b. d.m.v. aanroep sa length)	bp := bp + k of: rp := bp := bp + k of: ep := rp := bp + k	-
in sym	file nummer	ep - rp > 0?, is het symsoort?	bp := rp := rp + 1	volgend kar. uit file
inspect sym	file nummer	ep - rp > 0?, is het symsoort?	rp := rp + 1	"
unstack sym	file nummer	ep - bp > 0?, is het symsoort?	ep := ep - 1	"
in bin	"	ep - rp > 0?, is het bin soort?	bp := rp := rp + 1	"
inspect bin	"	ep - rp > 0?, is het bin soort?	rp := rp + 1	"
unstack bin	"	ep - bp > 0?, is het bin soort?	ep := ep - 1	"
out sym	file nummer, karakter	is het sym soort?	ep := ep + 1	-
out bin	file nummer, karakter	is het bin soort?	ep := ep + 1	-