stichting
mathematisch
centrum

MC

LR 2.4

september 1971

CONVERSIEPROGRAMMA'S

2e boerhaavestraat 49 amsterdam

0.1 Lijst van pagina's

Om na te gaan of een exemplaar al dan niet volledig is, kan gebruik worden gemaakt van onderstaande tabel.

Deze bladzijde zal bij elke aanvulling en/of wijziging worden vervangen.

0.2. Inhoudsopgave

CONVERSIEPROGRAMMA'S.

Hoofdstuk 1

Onderwerp :
conversie van algol 60 programma's van onderstreepstijl naar apostrofstijl.

Auteurs: J.V.M. van der Grinten en J. Kok.

Gebruiksaanwijzing :

invoer :

1 een getal t voor de conversie van tabulaties. de positie op de regel wordt op het volgende t-voud geplaatst. voor t = 8 blijft de lay-out ongewijzigd, maar bij een programma waarin alleen met tabulaties ingesprongen wordt kan nu de mate van inspringen gewijzigd worden.

2 een getalscheider.

3 a een getal w voor het ponsen van identificatieletters en -cijfers. w = 0 is 8 spaties in de laatste kolommen.

b een getalscheider.

c als w > 0 :
een apostrof, gevolgd door 8 symbolen voor de identificatie-kolommen, gevolgd door een twnr. opeenvolgende cijfers, tot en met het w-de symbool, vormen het getal waarbij de kaartnummering begint.
de overige symbolen worden per kaart niet gewijzigd. voor w = 8 verspringt het cijfer in kolom 80 met 1 per kaart, voor w = 7 verspringt het cijfer in kolom 79, enz.
voor kaartprogramma's die in MCALL moeten draaien en voor data wordt geadviseerd w = 0 te geven.

4 een algol tekst. bij conversie van meerdere programma's in 1 run moet achter het laatste end van elk programma een semicolon volgen om te voorkomen dat het begin van een nieuw programma als comment wordt vertaald.

in verband met de eventuele toevoeging van kaartnummering wordt afgeraden ook data mee te converteren.

Gebruikte procedures :
resym, read, csym, prsym, stringsymbol (zie LR 1.1).

Voorbeeld van een aanroep :

invoer :

```
8,8
'ZERO-000
```

real procedure ZERO(x,a,b,fx,e); value a,b; real x,a,b,fx; array e;

uitvoer :

'real' 'procedure' zero(x,a,b,fx,e); 'value' a,b; 'real' x,a,b,$zero-000
fx; 'array' e; zero-001

Methode en prestaties :

uitvoer :
het geconverteerde programma op ponskaarten. de eerste 72 kolommen worden voor het programma gebruikt, te lange regels worden met een dollarteken afgebroken.
de kolommen 73 - 80 worden volgens de invoer ad 3 gevuld.
de regeldrukker afdruk geeft precies het kaartbeeld.

de zin en mogelijke werking van een programma kunnen slechts gewijzigd worden als er strings geconverteerd worden. na een apostrof in string of comment wordt zo nodig een spatie toegevoegd.
geneste strings zijn mogelijk. niet voorzien wordt in het onderscheiden van identifiers die op case definitie na identiek zijn.
operatoren die een onderstreping of balk bevatten worden door een alternatieve representatie vervangen. de maximale lengte van word delimiters is 15 . een tabulatie wordt vervangen door 2 tot t+1 spaties (voor t zie invoer) behalve na een word delimiter.
in comment wordt punt gevolgd door komma vervangen door semicolon, verder vindt in string en comment geen conversie plaats.

Hoofdstuk 2

Onderwerp :
conversie van apostrofstijl naar onderstreepstijl.

Auteur: J. Kok.

Gebruiksaanwijzing :
invoer :
1 een getal m dat aangeeft met hoeveel spaties per begin-end er ingesprongen wordt. met m = 1 worden alle spaties letterlijk gekopieerd, maar met bijvoorbeeld m = 5 wordt met 5-vouden ingesprongen en worden zo afwijkingen tot 2 spaties ernaast gecorrigeerd. alleen aan het begin van regels worden 8 spaties door een tabulatie vervangen.
2 een programma (op kaarten. daar card image verwacht wordt moet voor toepassing op programma's op ponsband de procedure rsym veranderd worden). het programma begint met de eerste nlcr na de getalscheider na het getal ad 1.
3 zo veel extra blanke kaarten, dat de laatste regel van het programma als volledig beschouwd kan worden. output wordt namelijk gegeven zodra een symbool nlcr verschijnt, maar als een kaart een dollarteken bevat, wordt de rij symbolen van de volgende kaart niet begonnen met een symbool nlcr.

Gebruikte procedures :
prsym, pusym, runout, carriage, printtext, space, nlcr, absfixt, tab, stringsymbol, read, resym (zie LR 1.1).

Voorbeeld van een aanroep :

invoer :

```
8
        'real''procedure' zero(x,a,b,fx,e); 'value' a,b; 'real' x,a,b,f$zero-000
x; 'array' e;                                                          zero-001
```

uitvoer :

```
        real procedure zero(x,a,b,fx,e); value a,b; real x,a,b,fx; array e;
```

Methode en prestaties :

uitvoer :
de geconverteerde tekst op ponsband.

in strings vindt alleen conversie van stringhaken plaats.
in strings mogen geen stringhaken van de onderstreepstijl staan.
in end comment wordt na een reeks beginnend met '
gevolgd door en of el een sluitapostrof verwacht, terwijl
die reeks (van alleen letters en spaties) dan ontdaan
wordt van apostrofs en onderstreept wordt.
in comment wordt punt gevolgd door komma als einde
comment opgevat behalve als het aantal punten dat voor de
komma staat even is.
de volgende gespecificeerde conversie geschiedt alleen
buiten strings en comment :

apostrofwoord	⟶	onderstreept woord
de meest gebruikelijke afkortingen van apostrofwoorden behorend bij operatoren	⟶	die operatoren (te weten: eq, eqv, ge, gt, imp, le, lt, ne, not)
..	⟶	:
.=	⟶	:=
. gevolgd door ,	⟶	semicolon
'/'	⟶	:
××	⟶	↑
(/	⟶	[
/)	⟶	]

tussen word delimiters wordt zonodig een spatie geplaatst.

1. Onderwerp.

Een programma voor het comprimeren van ALGOL 60-teksten tot een minimaal aantal ponsingen.

2. Gebruiksaanwijzing.

2.1. Invoer.

De gehele invoer (dus inclusief de punten 2.1.1. t/m 2.1.2.2.) wordt gelezen alsof het commentaar uit een ALGOL 60-programma was, d.w.z. dat, als de invoer op kaarten staat, alleen de eerste 72 kolommen meetellen en de dollarteken-conventie geldt. De invoer moet bestaan uit de volgende punten.

2.1.1. Een lijst van 'beschermde namen', gescheiden door komma's en afgesloten door een puntkomma.
(N.B.: Gezien de eigenschappen van de gebruikte naamlijstalgoritme is het met het oog op de efficientie niet raadzaam de namen in alfabetische volgorde op te geven (zie voorbeeld in 4.2.)).

2.1.2. Een aanduiding van de vorm van uitvoer, en wel

2.1.2.1.
de letter 'c' voor uitvoer op kaarten of
de letter 'f' voor uitvoer op band in flexowriter-code of
de letter 'i' voor uitvoer op band in ISO-code of
de letter 't' voor het onderdrukken van ponsuitvoer.

2.1.2.2.
Als uitvoer op ponskaarten gevraagd is, moeten nog vier symbolen (geen spaties) volgen, die als identificatie geponst worden in kolom 73 t/m 76 van elke kaart.

2.1.3. Het te comprimeren programma. Van dit programma wordt verwacht dat het een correct en volledig ALGOL 60-programma is.

2.2. Uitvoer.

2.2.1. Ponsuitvoer.
De ponsuitvoer bestaat uit een ALGOL 60-programma, waaruit alle layout verwijderd is (behalve in strings), alle commentaar verwijderd is, (na comment, na end en 'vette komma'), alle standaard symbolen vervangen zijn door hun kortste representatie in apostrofstijl, en alle identifiers die niet als 'beschermd' waren opgegeven (zie 2.1.1.), vervangen zijn door andere, zo kort mogelijke, identifiers.
Als de ponsuitvoer op kaarten is, wordt in kolom 73 t/m 76 de identificatie geponst, terwijl in kolom 77 t/m 80 een met 1 oplopende nummering geponst wordt, te beginnen met 0001.

2.2.2. Regeldrukkeruitvoer.

2.2.2.1.

Indien de invoer niet aan de gestelde eisen voldoet, wordt een foutmelding gegeven. Er zijn twee soorten foutmeldingen.
De eerste soort wordt geconstateerd door de inleesroutine; de foutmelding heeft hetzelfde formaat als die van de ALGOL 60-vertaler en de foutnummers hebben dezelfde betekenis. Als foutnummers kunnen optreden 100, 101 en 108. Na zo'n fout wordt de verwerking voortgezet.
De tweede soort wordt geconstateerd door het compressiegedeelte; de foutmelding bestaat nu uit een mededeling, gevolgd door het symbool waarop de fout geconstateerd werd, gevolgd door de volgende 1000 symbolen uit de invoer, voor zover aanwezig. Na zo'n fout wordt de verwerking afgebroken.
Indien een foutmelding optreedt, is de geproduceerde ponsuitvoer vrijwel altijd onbruikbaar.

2.2.2.2.

Na afloop van de conversie wordt een lijst van identifiers met hun vervangers afgedrukt. Als de vervanger aangeduid wordt als xxx, werd de onderhavige identifier door zichzelf vervangen.
Deze lijst wordt altijd geproduceerd, ook als de verwerking ten gevolge van een fout afgebroken werd. Bij gebruik van de letter 't' als uitvoercode is het enige effect de productie van deze lijst.

2.3. Restricties.

Het gecomprimeerde programma heeft dezelfde werking als het oorspronkelijke, tenzij

2.3.1. er een globale naam uit de lijst met 'beschermde namen' ontbrak,
2.3.2. het programma in onderstreepstijl een string bevatte, die een opeenvolging van symbolen bevatte die in apostrofstijl als stringquote opgevat wordt (bij voorbeeld, ‡"‡ wordt gecomprimeerd tot """").
2.3.3. een van de beschermde namen gebruikt is als vervanger. Bij voorbeeld, als een programma meer dan 648 verschillende, niet-beschermde identifiers bevat, wordt de 649-ste vervangen door 'ln'; komt nu 'ln' voor in de lijst van beschermde namen, dan wordt ook de echte 'ln' vervangen door 'ln', waardoor een conflict van identifiers kan ontstaan.

3. Gebruikte procedures.
Het programma gebruikt de volgende procedures uit het Milli-systeem (zie LR 1.1.):

abs	new page	pusym
absfixt	nlcr	resymbol
carriage	out flexo	space
csym	out iso	stringsymbol
entier	print pos	tab
exit	print text	
line number	prsym	

4. Voorbeeld.

4.1. De invoer:

```
print, arctan; c infp
begin   comment this program prints an
                infinite sequence of values of pi;
infinite loop:
        print(4 times arctan(1));
        goto infinite loop
end
```

levert als ponsuitvoer:

'bgn'a:print(4×arctan(1));'goto'a'end' infp0001

en als regeldrukkeruitvoer:

```
×××     print
×××     arctan
a       infiniteloop
```

.

4.2. Het compressieprogramma zelf werd gecomprimeerd met de volgende invoer:

resymbol, csym, prsym, pusym, printtext, exit, outflexo, outiso, abs, entier, printpos, carriage, absfixt, space, linenumber, newpage, string symbol, nlcr, tab; c cmpr

(zie de opmerking over de volgorde van de identifiers in 2.1.1.).

5. Methode en prestatie.

5.1. Methode.
Het ALGOL 60-programma wordt ingelezen door de procedures van het procedurepakket 'basic symbols' (zie NR 18/71); deze procedures dragen zorg voor het verwijderen van commentaar en waarborgen dat het programma opgevat wordt in overeenstemming met de specificaties voor ALGOL 60-programma's voor het Milli-systeem (zie LR 1.1.).
Alle identifiers worden opgenomen in een naamlijst met behulp van een naamlijstalgoritme ontleend aan de CDL-compiler (zie MR 127/71). De namen worden genummerd in volgorde van binnenkomst, tenzij ze beschermd zijn, in welk geval ze het nummer -1 krijgen.
Bij uitvoer worden de symboolwaarden door de procedure 'algsym bs' (zie LR 2.4.4.) omgezet in opeenvolgingen van ponsbare symbolen. Als een naam niet het nummer -1 heeft, wordt uit het nummer een korte identifier geconstrueerd.

5.2. Prestaties.

De omvang van een normaal geschreven programma op kaarten wordt met een factor 4 tot 5 verkleind; de tijd benodigd voor compilatie van het gecomprimeerde programma is 60 tot 70 procent van de tijd benodigd voor het oorspronkelijke programma.
Het comprimeren kost ongeveer 1 milliuur per 250 ALGOL 60-symbolen.

1. Onderwerp.

Procedures voor het construeren van ALGOL 60-symbolen uit hun symboolwaarden zoals gedefinieerd in LR 1.1., paragraaf 6.1.1..
Voor het omgekeerde proces, het verkrijgen van de symboolwaarden van ALGOL 60-symbolen, zie "Handleiding bij het procedurepakket 'basic symbols'", NR 18/71.

2. Gebruiksaanwijzing.

Er zijn vier, min of meer analoge procedures beschikbaar, geheten 'algsym bl', 'algsym bs', 'algsym ul' en 'algsym us'.

2.1.
```
procedure algsym bl(sym, out); value sym; integer sym;
          procedure out;
comment bl = bold, long;
```

De integer 'sym' wordt opgezocht in de lijst gegeven in LR 1.1., paragraaf 6.1.1., het bijbehorende symboool in de kolom "apostrof-stijl voorkeursnotatie" wordt in ponsbare eenheden ontleed, en vervolgens wordt de procedure 'out' zo vaak als nodig is aangeroepen, met de resym-waarden van de ponsbare eenheden achtereenvolgens als parameter.
Zo resulteert de aanroep 'algsym bl(73,out)' in de aanroepen 'out(126); out(72)' en de aanroep 'algsym bl(105,out)' in 'out(120); out(14); out(23); out(13); out(120)'. Een uitzondering wordt gemaakt voor de stringquotes welke beide een aanroep van 'out(121)' produceren.
Elke symboolwaarde die niet in de lijst voorkomt, wordt ongewijzigd aan de procedure 'out' doorgegeven.

2.2.
```
procedure algsym bs(sym, out); value sym; integer sym;
          procedure out;
comment bs = bold, short;
```

De werking is analoog aan die van 'algsym bl'; in plaats van de voorkeursnotatie wordt de kortste combinatie van ponsingen genomen die door het Milli-systeem herkend wordt. Zo produceert 111 'ar' en niet 'array' en 113 'swit' en niet 'switch'.

2.3.
```
procedure algsym ul(sym, out, und); value sym; integer sym;
          procedure out; boolean und;
comment ul = underlined, long;
```

De werking is analoog aan die van 'algsym bl', in zoverre dat de "voorkeursnotatie in onderstreepstijl" geproduceerd wordt.

Een complicatie treedt op omdat tussen sommige symbolen een spatie geproduceerd moet worden; een combinateie als true= wordt door het Milli-systeem niet geaccepteerd, de combinatie true = wel. De procedure 'algsym ul' houdt in de boolean 'und' bij, of het laatst geproduceerde symbool onderstreept was, en voegt, indien nodig, voor het volgende symbool een spatie toe. Als parameter moet een boolean meegegeven worden, die voor de eerste aanroep op false geinitialiseerd moet worden.

2.4. procedure algsym us(sym, out, und); value sym; integer sym;
procedure out; boolean und;
comment us = undelined, short;

De werking is analoog aan die van 'algsym ul'; in plaats van de voorkeursnotatie wordt de kortste combinatie van ponsingen genomen die door het Milli-systeem herkend wordt. Zo produceert 111 ar en niet array en 113 swit en niet switch.

3. Gebruikte procedures.
Alle procedures gebruiken de procedure 'string symbol' uit het Milli-systeem (zie LR 1.1.).

4. Voorbeelden.

4.1. Het programma

```
begin   boolean b; b:= false;
        algsym ul(104, prsym, b);
        algsym us(112, prsym, b)
end
```

geeft als regeldrukkeruitvoer

begin proc

4.2. Als 'nxt bsc sbl' een integer procedure is die bij herhaald aanroepen de symboolwaarden van de symbolen van een ALGOL 60-programma aflevert, wordt dit programma geponst door de compound statement

begin 1: algsym bs(nxt bsc sbl, pusym); goto 1 end

5. Prestaties.

Een aanroep kost gemiddeld ruwweg 1 milli-seconde per geproduceerde ponsing.

6. De ALGOL 60-teksten.

6.1.

```
'proc' algsym bl(sym, out); 'val' sym; 'int' sym; 'proc' out;
'begin' 'integer' p, i, j;

  'if' sym < 68 'then' out(sym) 'else'
  'if' sym < 81 'then'
  'begin'
    p:= stringsymbol(sym-68, "ub b u u uu  ");
    'if' p = 30 'then' out(126) 'else'
    'if' p = 11 'then' out(127);
    out(stringsymbol(sym-68, ":Λ==<<>>⌉=⌈VΛ"))
  'end' 'else'
  'if' sym = 92 'then' 'begin' out(90);  out(70)  'end' 'else'
  'if' sym=102 V sym=103 'then' out(121) 'else'
  'if' sym > 117 'then' out(sym) 'else'
  'begin'
    p:= stringsymbol(sym - 81, "012345,,,,,,,6789,,,,,,abcdefghijklmn");
    'if' p = 87 'then' out(sym) 'else'
    'begin' out(120);
      p:= p × 9 - 1;
      'for' i:= 1 'step' 1 'until' 9 'do'
      'begin'   j:= stringsymbol(i + p, "$
goto     for      step     until    while    do       $
if       then     else     comment  begin    end      $
own      real     integer  boolean  string   array    $
procedureswitch   label    value    true     false    ");
        'if' j 'ne' 93 'then' out(j)
      'end';
      out(120)
    'end'
  'end';

'end' algsym bl;
```

6.2.

```
'proc' algsym bs(sym, out); 'val' sym; 'int' sym; 'proc' out;
'begin' 'integer' p, i, j;

  'if' sym < 68 'then' out(sym) 'else'
  'if' sym < 81 'then'
  'begin'
    p:= stringsymbol(sym-68, "ub b u u uu  ");
    'if' p = 30 'then' out(126) 'else'
    'if' p = 11 'then' out(127);
    out(stringsymbol(sym-68, ":Λ==<<>>⌉=⌈VΛ"))
  'end' 'else'
  'if' sym = 92 'then' 'begin' out(90);  out(70)  'end' 'else'
  'if' sym=102 V sym=103 'then' out(121) 'else'
  'if' sym > 117 'then' out(sym) 'else'
```

```
    'begin'
      p:= stringsymbol(sym - 81, "012345,,,,,,,6789,,,,,,abcdefghijklmn");
      'if' p = 87 'then' out(sym) 'else'
      'begin' out(120);
        p:= p × 9 - 1;
        'for' i:= 1 'step' 1 'until' 4 'do'
        'begin'   j:= stringsymbol(i + p, "$
goto     for      step     until    while    do       $
if       then     else     co       bgn      end      $
own      real     int      boolean  string   ar       $
procedureswitch   label    val      true     false    ");
          'if' j 'ne' 93 'then' out(j)
        'end';
        out(120)
      'end'
    'end';

  'end' algsym bs;
```

6.3.

```
  'proc' algsym ul(sym, out, und); 'val' sym;
    'int' sym; 'proc' out; 'bool' und;
  'begin'   'integer' p, i, j;  'boolean' und2;

    und2:= 'false';
    'if' sym < 68 'then' out(sym) 'else'
    'if' sym < 81 'then'
    'begin'
      'if' und 'then'
      'begin'
        'if' stringsymbol(80 - sym, "0011010100001") = 1 'then' out(93)
      'end';
      p:= stringsymbol(sym-68, "ub b u u uu  ");
      'if' p = 30 'then' out(126) 'else'
      'if' p = 11 'then' out(127);
      out(stringsymbol(sym-68, ":Λ==<<>>⌉=⌉∨Λ"))
    'end' 'else'
    'if' sym = 92 'then' 'begin' out(90);  out(70)  'end' 'else'
    'if' sym = 102 'then' 'begin' out(127); out(72) 'end' 'else'
    'if' sym = 103 'then' 'begin' out(127); out(74) 'end' 'else'
    'if' sym > 117 'then' out(sym) 'else'
    'begin'
      p:= stringsymbol(sym - 81, "012345,,,,,,,6789,,,,,,abcdefghijklmn");
      'if' p = 87 'then' out(sym) 'else'
      'begin'
        'if' und 'then' out(93); und2:= 'true';
        p:= p × 9 - 1;
        'for' i:= 1 'step' 1 'until' 9 'do'
        'begin'   j:= stringsymbol(i + p, "$
goto     for      step     until    while    do       $
if       then     else     comment  begin    end      $
```

```
own      real     integer  boolean  string   array    $
procedureswitch   label    value    true     false    ");
          'if' j 'ne' 93 'then'
          'begin'  out(126);  out(j)  'end'
        'end'
      'end'
    'end';

    und:= und2
  'end' algsym ul;
```

6.4.

```
  'proc' algsym us(sym, out, und); 'val' sym;
    'int' sym; 'proc' out; 'bool' und;
  'begin'   'integer' p, i, j;  'boolean' und2;

    und2:= 'false';
    'if' sym < 68 'then' out(sym) 'else'
    'if' sym < 81 'then'
    'begin'
      'if' und 'then'
      'begin'
        'if' stringsymbol(80 - sym, "0011010100001") = 1 'then' out(93)
      'end';
      p:= stringsymbol(sym-68, "ub b u u uu  ");
      'if' p = 30 'then' out(126) 'else'
      'if' p = 11 'then' out(127);
      out(stringsymbol(sym-68, ":∧==<<>>⌉=⌉∨∧"))
    'end' 'else'
    'if' sym = 92 'then' 'begin' out(90);  out(70)  'end' 'else'
    'if' sym = 102 'then' 'begin' out(127); out(72) 'end' 'else'
    'if' sym = 103 'then' 'begin' out(127); out(74) 'end' 'else'
    'if' sym > 117 'then' out(sym) 'else'
    'begin'
      p:= stringsymbol(sym - 81, "012345,,,,,,,6789,,,,,,,abcdefghijklmn");
      'if' p = 87 'then' out(sym) 'else'
      'begin'
        'if' und 'then' out(93); und2:= 'true';
        p:= p × 9 - 1;
        'for' i:= 1 'step' 1 'until' 4 'do'
        'begin'  j:= stringsymbol(i + p, "$
goto     for      step     until    while    do       $
if       then     else     co       bgn      end      $
own      real     int      boolean  string   ar       $
procedureswitch   label    val      true     false    ");
          'if' j 'ne' 93 'then'
          'begin'  out(126);  out(j)  'end'
        'end'
      'end'
    'end';
```

```
    und:= und2
'end' algsym us;
```