

PASCAL BLS.

Brugermanual

Programmeringsmanual

KOLDSTART

START: E2180 aaaa

WARMSTART. E2182.

Blue Label Software Pascal

BRUGERMANUAL

TAPE 1000-3F00

1400-3E00

4 F F F

PolyData
microcenter

Strandboulevarden 63
2100 København Ø • Tlf. (01) 42 07 05

INDEX

0. INTRODUKTION	2
1. KONTROLPROGRAMMET	3
2. IND- OG UDLÆSNING AF PROGRAMTEKST	3
2.1 SAVE kommandoer	3
2.2 LOAD kommandoer	4
2.3 VERIFY kommandoer	4
3. EDITOREN	4
3.1 Editeringskommandoer	5
3.2 Cursorkommandoer	6
3.3 Blokkommandoer	6
3.4 Søgekommandoer	7
3.5 Tabulatorkommandoer	7
3.6 Andre editorkommandoer	7
4. COMPILEREN	8
4.1 COMPILE kommandoer	9
4.2 RUN kommandoer	9
4.3 TAPE kommandoer	9
4.4 FIND kommandoer	10
5. ANDRE KONTROLKOMMANDOER	11
5.1 MEMORY kommandoer	11
5.2 ZAP kommandoer	11
5.3 QUIT kommandoer	11

APPENDIX

- A. Indlæsning og opstart af systemet
- B. Systemarbejdslageret
- C. Memory maps
- D. Den brugerdefinerbare udskriftsrutine
- E. Kommandooversigter

BLS Pascal er et komplet 12 K sprogsystem udviklet til NASCOM mikrodatamaten. Systemet indeholder et kontrolprogram, en skærmeditor, en pascalcompiler og en runtime pakke med de fornødne runtime rutiner.

Compileren behøver kun et gennemløb, og producerer direkte Z-80 maskinkode. Det oversatte program kræver kun tilstedeværelse af runtime pakken for at kunne køre.

Denne manual beskriver systemets opbygning og betjening. Selve programmeringssproget er defineret i BLS Pascal programmeringsmanual.

BLS Pascal systemet og de dertil hørende manualer er udviklet af Anders Hejlsberg i efteråret 1980.

1: KONTROLPROGRAMMET

Når systemet er indlæst og opstartet som beskrevet i APPENDIX A, udskrives der på skærmen:

```
BLS Pascal version x.x
Copyright (C) 1981
Poly-Data microcenter ApS
>_
```

hvor x.x er versionsnummeret. Vinkelen er systemets prompt-tegn, og indikerer at en kontrolkommando kan indtastes. Under indtastning af kontrolkommandoer kan følgende specialnøgler bruges:

<BS>	Sletter det sidst indtastede tegn.
<ESC>	Sletter hele indtastningen.
<ENTER>	Accepterer indtastningen.

En kontrolkommando består af et kommandoord eventuelt efterfulgt af en parameter. Der skal være mindst et blanktegn mellem parametren og kommandoordet. Kommandoer kan forkortes til det første bogstav i kommandoordet. Således gælder der at kommandolinien:

LOAD testprogram

Svarer til:

L testprogram

Der er elleve kontrolkommandoer i BLS Pascal systemet. Disse kan, efter deres funktion, opdeles i 4 grupper:

1. Ind- og udlæsning af programtekst.
2. Editoren.
3. Compileren.
4. Andre funktioner

2: IND- OG UDLÆSNING AF PROGRAMTEKST

Ind- og udlæsning af programtekst til og fra båndoptager varetages af de 3 kontrolkommandoer SAVE, LOAD og VERIFY. Formatet af en udlæsning svarer til NAS-SYS W-format, hvilket betyder, at hvis man under en indlæsning får en checksumfejl i en blok, kan man spole tilbage og indlæse den påny.

2.1 SAVE kommandoen

Denne kommando bruges til udlæsning af programtekster. Formatet af kommandolinien er:

SAVE filnavn eller S filnavn

hvor filnavn er det navn, man ønsker programmet gemt under. Filnavne kan have hvilkensomhelst længde, og må gerne indeholde blanktegn.

2.2 LOAD kommandoen

Denne kommando bruges til indlæsning af programtekster. Formatet af kommandolinien er:

LOAD filnavn eller L filnavn

hvor filnavn er navnet på det program, man ønsker at indlæse. Hvis filnavnet udelades, indlæses den første fil, der mødes. Hvergang et program bliver fundet, udskrives der:

File filnavn found

Det indlæste program bliver lagt i lageret efter det nuværende. Således er der mulighed for at sammensætte et program af i forvejen bestående underprogrammer. Ønsker man at indlæse et program for sig selv, bør den nuværende programtekst først slettes, fx. med ZAP kommandoen. LOAD kommandoen kan på ethvert tidspunkt afbrydes ved et tryk på <ESC>. Ved kald af LOAD kommandoen slettes objekt-koden.

afbryder LOAD

2.3 VERIFY kommandoen

Denne kommando bruges til at verificere en udlæsning af en programtekst. Formatet af kommandolinien er:

VERIFY filnavn eller V filnavn

hvor filnavn er navnet på den programtekst, man ønsker at verificere. Hvis filnavnet udelades, verificeres den første fil, der mødes. VERIFY kommandoen kan på ethvert tidspunkt afbrydes ved et tryk på <ESC>.

3: EDITOREN

Editoren i BLS Pascal systemet er skærmorienteret. Dette betyder bl. a., at der ikke, i modsætning til BASIC, er behov for linienumre i programteksten.

En skærmorienteret editor virker skærmen som et vindue, der kan flyttes rundt over programteksten. Cursoren befinder sig altid inden for dette vindue, og angiver, ved sin position, hvor der skal indsættes, fjernes eller rettes nogle tegn eller linier.

I editoren kan programlinier være op til 80 tegn lange. Dette betyder, at skærmen, ud over at kunne flyttes frem og tilbage i programteksten, også kan flyttes til siderne. Hvis man skriver over 48 tegn på en linie, vil cursoren ikke flytte ned på næste linie, men skærmen vil i stedet scrolle mod venstre, således at de forreste tegn på linien 'forsvinder'. Først når der er skrevet 80 tegn, flytter cursoren ned på næste linie.

Umiddelbart kan denne opbygning virke forvirrende, men i Pascal er det meget ofte nødvendigt med linier der er længere end 48 tegn, for at programtekstens overskuelighed skal bevares.

editoren aktiveres med kommandoen:

EDIT eller E

Ved indgangen kommer man tilbage til det sted i teksten, man forlod ved sidste editering. Hvis det er første gang editoren aktiveres, eller hvis ZAP kommandoen har været i brug, slettes skærmen, og cursoren placeres i øverste venste hjørne.

Editoren har 27 kommandoer, der har ASCII-værdierne mellem 1 og 27 (dvs. kontroltegn). Alle øvrige tegn (inklusive grafiktegn) vil, når de indtastes, blive indsat i programteksten på det sted hvor cursoren befinder sig, og denne vil derefter, hvis det er muligt, flytte en position mod højre.

Under brug af editoren kan det ske, at arbejdslageret bliver opbrugt. I så tilfælde slettes skærmen, og der udskives:

Overflow

Brugeren må da, hvis det er muligt, udvide arbejdslageret ved at flytte MTOP til en højere adresse (se APPENDIX B).

Under beskrivelsen af editorens kommandoer bruges de følgende notationer:

CTRL/ eller SHFT/ efterfulgt af et bogstav betyder, at bogstavet indtastes samtidig med at control- eller shift-nøglen holdes nede.

HP betyder højrepil, VP venstrepil, NP nedpil og OP oppil.

3.1 Editeringskommandoer

Editeringskommandoerne er de kommandoer der retter i programteksten.

<BS>	Flytter cursoren en position mod venstre og sletter det tegn, den bliver placeret oven i. Hvis cursoren står i første position på en linie flyttes den til position nummer 79 i den foregående linie.
<ENTER>	Flytter cursoren en linie ned og indsætter en tom linie. Cursoren placeres i første position af denne linie.
<ESC>	Sletter den linie cursoren står på og flytter cursoren en linie op. Cursoren flytter til første position.
SHFT/HP CTRL/V	Flytter den del af linien, der står efter cursoren, en position mod højre og indsætter et blanktegn.
SHFT/VP CTRL/U	Fjerner det tegn, der står under cursoren, og rykker resten af linien en position mod venstre.
SHFT/NP CTRL/Z	Flytter den linie, cursoren står på, og de understående linier en linie ned og indsætter en tom linie. Cursoren flytter til første position.
SHFT/OP CTRL/Y	Sletter den linie, cursoren står på, og flytter de understående linier en linie op. Cursoren flytter til første position.

3.2 Cursorkommandoer

Cursorkommandoerne er kommandoer, der flytter cursoren uden at rette i programteksten.

HP CTRL/R	Flytter cursoren een position mod højre. Hvis cursoren står i position 79, flytter den til første position i næste linie.
VP CTRL/Q	Flytter cursoren een position mod venstre. Hvis cursoren står i første position flytter, den til position 79 i den forrige linie.
NP CTRL/T	Flytter cursoren een linie ned. Hvis cursoren står i nederste linie, scroller skærmen een linie op.
OP CTRL/S	Flytter cursoren een linie op. Hvis cursoren står i øverste linie, scroller skærmen een linie ned.
CTRL/B	Flytter cursoren til første position i programtekstens første linie.
CTRL/E	Flytter cursoren til første position i programtekstens sidste linie.
CTRL/N	Flytter cursoren 14 linier frem i programteksten.
CTRL/O	Flytter cursoren 14 linier tilbage i programteksten.
<LF> CTRL/J	Flytter cursoren til første position på den nuværende linie.
<CS> CTRL/L	Flytter cursoren til positionen umiddelbart efter det sidste tegn på den nuværende linie.

3.3 Blokkommandoer

Blokkommandoerne er kommandoer, der berører hele blokke af programteksten. En blok afmærkes ved hjælp af blokmærker, der kan sættes med CTRL/A kommandoen. En blokkommando berører kun den første afmærkede blok i programteksten. Hvis der ikke er nogen blokafmærkninger i programteksten, ignorerer blokkommandoerne (dog ikke CTRL/A).

CTRL/A	Denne kommando skal efterfølges af et bogstav. 'B' indikerer at man ønsker at sætte et start-blokmærke, og 'E' indikerer at man ønsker at sætte et slut-blokmærke. Blokmærker bliver altid indsat før den nuværende linies første position. Hvis der i forvejen står et blokmærke i linien, bliver CTRL/A kommandoen ignoreret.
CTRL/D	Fjerner den første mærkede blok (inclusive blokmærker) fra programteksten. Cursoren flytter til den første position i den linie, hvor slut-blokmærket stod.
CTRL/I	Indsætter den første mærkede blok (exclusive blokmærker) umiddelbart før den linie, cursoren står på. Hvis cursoren står på en af linierne mellem blokmærkerne, eller på den linie, hvor slut-blokmærket står, ignorerer CTRL/I kommandoen.

CTRL/P Udskriver den første mærkede blok via den bruger-definerbare udskriftsrutine (se APPENDIX D). Kommandoen CTRL/P skal efterfølges af et bogstav. Hvis dette er et 'L', udskrives blokken med linienumre.

3.4 Søgekommandoer

De to søgekommandoer bruges til at finde de steder i programteksten, hvor en indtastet søgetekst forekommer.

CTRL/F Finder den første forekomst af en søgetekst på max. 40 tegn. Idet CTRL/F indtastes, indsættes der en tom linie på skærmen, og som prompt-tegn udskrives der en højrepil. Herefter kan søgeteksten indtastes, under brug af de samme specialnøgler som ved kommandoindtastninger. Når indtastningen er afsluttet forsvinder linien igen. Hvis teksten bliver fundet under søgningen, bliver cursoren placeret oven i det første tegn i teksten. Hvis teksten ikke bliver fundet, bliver cursoren stående. Søgningen starter altid i linien umiddelbart efter den nuværende.

CTRL/C Fortsætter søgningen efter den sidst indtastede søgetekst.

3.5 Tabulatorkommandoer

CTRL/K Denne kommando bruges til at sætte tabulatorlængden. CTRL/K skal efterfølges af et bogstav, der afgør tabulatorlængden. Bogstavet 'A' svarer til en længde på 1, bogstavet 'B' til en længde på 2, etc., hvilket svarer til, at længden bliver bogstavets ASCII værdi minus 64. Den maksimale tabulatorlængde er 63. Hvis man vælger en tabulatorlængde på 0 (ved at trykke CTRL/K efterfulgt af '@'), virker tabulatoren anderledes, idet den nu flytter cursoren til den position i linien, der svarer til positionen af det første tegn i den foregående linie. Tabulatorlængden ændres ikke ved, at editoren forlades.

<CH> Flytter cursoren frem til den næste tabulatorposition, eller, hvis tabulatorlængden er 0, til den position i den nuværende linie, der svarer til positionen af det første tegn i den foregående linie.

3.6 Andre editorkommandoer

CTRL/G Denne kommando bruges til at ændre <GRAPH> nøglens funktion. CTRL/G skal efterfølges af et bogstav. Hvis man trykker 'A' vil <GRAPH> nøglen nu virke som en APLHA-LOCK nøgle: Hvergang den nedtrykkes, reverseres <SHIFT> nøglens funktion. Hvis CTRL/G efterfølges af et 'G', virker <GRAPH> nøglen igen som normalt. <GRAPH> nøglens funktion bliver husket selvom editoren forlades.

CTRL/X Sletter skærmen og går tilbage til kontrolprogrammet. CTRL/X fjerner desuden alle blokmærker fra programteksten.

4: COMPILEREN

Compileren er hjertet i sprogsystemet. Ved hjælp af denne kan programtekster oversættes (compileres) til objektkode.

Compileren kan arbejde på flere forskellige måder:

- 1) Med COMPILE/RUN kommandoerne gemmes den producerede objektkode direkte ud i lageret, umiddelbart efter programteksten. Denne metode er langt den hurtigste, men den stiller det krav, at der er plads til både sprogsystemet, programteksten og objektkoden i lageret på een gang.
- 2) Med TAPE kommandoen udlæses den producerede objektkode til båndoptager. Denne metode er naturligvis langsommere end den ovenstående, men den har den fordel, at der ikke kræves lagerplads til objektkoden. Desuden giver TAPE kommandoen brugeren mulighed for at adressere objektkoden hvorsomhelst i lageret.
- 3) Når compileren aktiveres via FIND kommandoen, genereres der ingen objektkode. FIND kommandoen bruges til at finde et sted i programteksten, der svarer til en bestemt adresse i objektkoden. Dette er især brugbart, hvis man under kørslen af et program får en runtime fejl og ønsker at finde den sætning i programteksten, der er skyld i fejlen.

Hvis man under en compilering får en fejl, bliver editoren automatisk aktiveret, og på skærmens øverste linie udskrives fejlsnummer. Cursoren placeres oven i eller umiddelbart efter den tekst, der er skyld i fejlen.

Lad os antage at man har indtastet det følgende program:

```
VAR i: INTEGER;
BEGIN
  readln(i); writeln('Kvadratet er ',sqr(j))
END.
```

Programmet indeholder en fejl, idet variabelen j ikke er erklæret. Hvis man prøver at compilere programmet, vil der ske følgende:

```
Compilation error 64                      Press <SPACE>
  readln(i); writeln('Kvadratet er ',sqr(j))
END.
```

Cursoren er placeret oven i j'et, på den første linie, for at angive fejlen. Idet man trykker <SPACE> forsvinder fejltteksten, og programmet kan editeres, som havde man aktiveret editoren på normal vis.

Under en compilering kan det ske, at alt arbejdslageret bliver opbrugt. Da udskrives der:

Overflow

Brugeren må nu, hvis det er muligt, udvide arbejdslageret på en af de følgende måder:

- 1) Hvis systemet ikke har tilgang til hele RAM-lageret, bør MEMTOP flyttes til en højere adresse (se herom i APPENDIX B).
- 2) Hvis compileren blev aktiveret med COMPILE eller RUN kommandoerne, kan man i stedet bruge TAPE kommandoen og udlæse objekt-koden på bånd.

4.1 COMPILE kommandoen

Når compileren aktiveres via COMPILE kommandoen, bliver den producerede objekt-kode lagt direkte ud i lageret umiddelbart efter programteksten. Formatet af kommandolinien er:

COMPILE eller C

Efter kommandokaldet udskrives der:

Compiling

Forudsat at der ikke er fejl i programteksten, bliver følgende uskrevet, efter at kompileringen er slut:

Compiling OK
Text: \$aaaa \$bbbb <xxxxx>
Code: \$cccc \$dddd <yyyyy>

hvor aaaa og bbbb er programtekstens start- og slutadresser (i hexnotation), og xxxxx er programtekstens størrelse. cccc, dddd og yyyyy er de tilsvarende parametre for objekt-koden.

4.2 RUN kommandoen

Denne kommando bruges, når man ønsker at udføre et program. Hvis programteksten ikke er compileret (ved hjælp af COMPILE eller en tidligere RUN kommando), vil dette blive gjort før programmet startes. Forudsat at der ikke opstår fejl under kompileringen, eller hvis programteksten i forvejen er compileret, udskrives der:

Running

hvorefter kontrollen bliver overgivet til programmet. Efter endt kørsel af programmet, bliver kontrollen tilbagegivet til sprogsystemet. Hvis der opstår en runtime fejl under kørslen af et program, udskrives der:

Runtime error xx at \$nnnn

hvorefter kontrollen overgives til sprogsystemet (eller til NAS-SYS hvis programmet er compileret med TAPE kommandoen). xx er fejlens nummer, og nnnn er fejladressen i hexnotation. Adressen er ikke den absolutte adresse, men offsetadressen fra programmets startadresse. Ved hjælp af FIND kommandoen (uden argument) kan man nu finde den sætning i programteksten, der er skyld i fejlen.

4.3 TAPE kommandoen

Når compileren aktiveres via TAPE kommandoen, bliver objekt-koden udlæst til båndoptageren; denne bør derfor startes før kommandokaldet. Formatet af kommandolinien er:

hvor nnnn er den hexadecimale adresse, man ønsker objekt-koden adresseret til. Hvis nnnn udelades, vælger systemet \$2180 som startadresse (hvilket er slutadressen for runtime pakken og startadressen for kontrolprogrammet). Objekt-koden udskrives i NAS-SYS W-format, og kan senere indlæses med NAS-SYS R-kommandoen. Når udlæsningen er slut, udskrives der:

xxxx End.

Hvor xxxx er objekt-kodens slutadresse.

Når programmet senere indlæses, kan det startes med NAS-SYS kommandoen Ennnn, hvor nnnn er startadressen. For at programmet kan køre, kræves det at runtime pakken er tilstede i lageret mellem \$1000 og \$2180; imidlertid behøver resten af sprogsystemet ikke at være tilstede. Når et program virker tilfredsstillende, kan man således udlæse objekt-koden med TAPE kommandoen, og derefter sammensætte denne med runtime pakken, for at danne en færdig objekt-kode, der kan køre helt for sig selv.

Eventuelle maskinkoderutiner kan med fordel placeres imellem \$2180 og programmets startadresse. Har man for eksempel en samling maskinkoderutiner der fylder 128 bytes (\$80), kan disse startes i \$2180 og programmet i \$2200.

4.4 FIND kommandoen

FIND kommandoen bruges til at finde et sted i programteksten, der svarer til en adresse i objekt-koden. Når compileren aktiveres via FIND, genereres der ingen objekt-kode. Formatet af kommandolinien er:

FIND nnnn

eller

F nnnn

hvor nnnn er den adresse man ønsker at finde. Adressen er ikke den absolutte adresse, men en offsetadresse fra programmets startadresse. Hvis et program fx. starter i \$2180, og man ønsker at finde det sted i programteksten, der svarer til adressen \$2295, skal man indtaste FIND 115. Hvis nnnn udelades bruges adressen på den sidst forekomne runtime fejl. Efter kommandokaldet udskrives der:

Searching

Hvis offsetadressen passer under søgningen, reagerer systemet ved at aktivere editoren og udskrive:

Compilation error 00

Press <SPACE>

Cursoren er placeret umiddelbart efter den relevante tekst. Idet man trykker <SPACE> forsvinder fejlmeldingen, og programmet kan editeres som normalt.

Hvis offsetadressen ikke passer, udskrives der:

Searching ?

Ved brugen af FIND kommandoen, slettes en eventuel objekt-kode.

5: ANDRE KONTROLKOMMANDOER

5.1 MEMORY kommandoen

Denne kommando oplyser start- og slutadresserne og størrelsen af programteksten, og de samme parametre for en eventuel objektkode. Formatet af kommandolinien er:

MEMORY

eller

M

Herved udskrives der:

Text: \$aaaa \$bbbb <xxxxx>

hvor aaaa og bbbb er programtekstens start- og slutadresser (i hexnotation) og xxxxx er programtekstens størrelse. Hvis programmets objektkode er tilstede, udskrives der tillige:

Code: \$cccc \$dddd <yyyyy>

hvor cccc, dddd og yyyyy er de tilsvarende parametre for objektkoden.

5.2 ZAP kommandoen

Denne kommando sletter programtekst og objektkode. Formatet af kommandolinien er:

ZAP

eller

Z

Pas godt på med denne kommando.

5.3 QUIT kommandoen

Denne kommando overgiver kontrollen til NAS-SYS. Formatet af kommandolinien er:

QUIT

eller

Q

Systemet kan senere varmstartes som beskrevet i APPENDIX A.

APPENDIX A -- INDLESNING OG OPSTRART AF SYSTEMET

BLS Pascal systemet leveres på et kassettebånd, der er udlæst med NAS-SYS W-kommando på 1200 baud. Indlæsningen foregår ved hjælp af NAS-SYS R-kommando.

BLS Pascal systemet koldstartes ved at skrive:

E1000 aaaa

eller

E2180 aaaa

hvor aaaa er den sidste adresse i RAM-lageret sprogsystemet må benytte. Hvis aaaa udelades benyttes alt tilgængeligt RAM-lager.

BLS Pascal systemet varmstartes ved kommandoen:

E2182

1000 → 3F 00
2F 00

Systemets bruger 128 bytes arbejdslager i området \$C80 til \$D00. I dette område kan følgende være af interesse for brugeren:

C80-C81	MTOP	Den øverste adresse i RAM-lageret sprogsystemet må bruge.
C82-C83	EOFP	Programtekstens slutadresse.
C84-C85	PEND	Objektkodens slutadresse.

D.1

APPENDIX D -- DEN BRUGERDEFINERBARE UDSKRIFTSROUTINE

Når man bruger CTRL/P kommandoen i editoren, bliver udskriften sendt via den brugerdefinerbare udskriftsrutine i NAS-SYS.

Adressen på denne rutine skal stå i adresserne \$C78-\$C79, og den bør følge de regler, der er givet i afsnittet INPUT AND OUTPUT i NAS-SYS manualen.

Under udviklingen af et program har systemets RAM-lager typisk det følgende layout:

0C80	+	-----	+
	!	systemets arbejdslager	!
0D00	+	-----	+
	!	systemets stack	!
1000	+	-----	+
	!	runtime pakke	!
2180	+	-----	+
	!	kontrolprogram/editor	!
2980	+	-----	+
	!	compiler	!
4000	+	-----	+
	!	programtekst	!
EOFP	+	-----	+
	!	objektkode	!
PEND	+	-----	+
	!	programmets arbejdslager	!
MTOP	+	-----	+
	!	evt. maskinkoderutiner	!
	+	-----	+

Under kørslen af det færdige program har systemets RAM-lager typisk det følgende layout:

0C80	+	-----	+
	!	systemets arbejdslager	!
0D00	+	-----	+
	!	systemets stack	!
1000	+	-----	+
	!	runtime pakke	!
2180	+	-----	+
	!	evt. maskinkoderutiner	!
	+	-----	+
	!	objektkode	!
	+	-----	+
	!	programmets arbejdslager	!
MTOP	+	-----	+

Blue Label Software
Pascal

PROGRAMMERINGSMANUAL

Folytata
microcenter

Strandboulevarden 63
2100 København Ø • Tlf. (01) 42 07 05

0. INTRODUKTION	3
1. SPROGETS GRUNDLEGGENDE ELEMENTER	4
1.1 Symboler	4
1.2 Reserverede ord og standardidentifiere	4
1.3 Seperatorer	4
2. BRUGERDEFINEREDE SPROGELEMENTER	5
2.1 Identifiere	5
2.2 Tal	5
2.3 Strengte	5
2.4 Kommentarer	5
3. DATATYPER	6
3.1 Integers	6
3.2 Reals	6
3.3 Booleans	6
3.4 Strings	6
3.5 Arraystrukturer	6
3.5.1 mem arraystrukturen	7
4. ERKLÆRINGER OG DEFINERINGER	8
4.1 Labelerklæringer	8
4.2 Konstantdefineringer	8
4.3 Variabelerklæringer	9
4.4 Procedure- og funktionserklæringer	9
5. UDTRYK	10
5.1 NOT operatoren	10
5.2 Multipliserende operatorer	10
5.3 Adderende operatorer	10
5.4 Sammenlignende operatorer	10
5.5 Paranteser i udtryk	11
5.6 Funktionskald	11
6. SÆTNINGER	12
6.1 Simple sætninger	12
6.1.1 Tilskrivningssætningen	12
6.1.2 Proceduresætningen	12
6.1.3 GOTO-sætningen	12
6.1.4 INIT-sætningen	12
6.1.5 Tomme sætninger	13
6.2 Strukturerede sætninger	14
6.2.1 Sammensat sætning	14
6.2.2 Betingede sætninger	14
6.2.2.1 IF-sætningen	14
6.2.2.2 CASE-sætningen	14
6.2.3 Repetitionssætninger	15
6.2.3.1 WHILE-sætningen	15
6.2.3.2 REPEAT-sætningen	15
6.2.3.3 FOR-sætningen	15
7. PROCEDURER	17
7.1 Procedureerklæringer	17
7.1.1 Procedurehovedet	17
7.1.2 Erklæringsdelen	17
7.1.3 Sætningsdelen	17
7.2 Standardprocedurer	17

8. FUNKTIONER	19
8.1 Funktionserklæringer	19
8.1.1 Funktionshovedet	19
8.1.2 Erklæringsdelen	19
8.1.3 Sætningsdelen	19
8.2 Standardfunktioner	20
8.2.1 Reelle funktioner	20
8.2.2 Heltalsfunktioner	20
8.2.3 Stringfunktioner	20
8.2.4 Konverteringsfunktioner	21
8.2.5 Andre standardfunktioner	21
9. PARAMETRE	22
9.1 Parametertyper	22
9.1.1 Value-parametre	22
9.1.2 Var-parametre	22
9.2 Formelle og aktuelle parametre	22
10. INDLÆSNING OG UDLÆSNING	23
10.1 Indlæsning fra konsollen	23
10.1.1 read proceduren	23
10.1.2 readln proceduren	23
10.2 Udlæsning til konsollen	24
10.2.1 write proceduren	24
10.2.2 writeln proceduren	25
10.3 Indlæsning og udlæsning af arraystrukturer ...	25
10.3.1 save proceduren	25
10.3.2 load proceduren	25

APPENDIX

- A. Syntaxdiagrammer
- B. Eksempler på underprogrammer
- C. Systemarbejdslageret
- D. Intern datarepræsentation
- E. Eksterne underprogrammer
- F. Benchmarktests og tider
- G. Litteraturliste
- H. Compiler fejlmeddelelser
- I. Runtime fejlmeddelelser

0: INTRODUKTION

Hovedmålet med udviklingen af BLS Pascal systemet er at skabe et alternativ til BASIC programmeringssproget.

Programmer skrevet i BASIC har en tendens til at blive uoverskuelige og kører ofte meget langsomt. Programmer skrevet i det blokstrukturerede sprog Pascal er langt mere overskuelige og kører meget hurtigere, da de ikke skal fortolkes, men i stedet oversættes til maskinkode.

BLS Pascal er et meget kompakt system, der kun kræver ca. 12 K lager, hvoraf de ca. 5 1/2 K er compilerdelen. Naturligvis kan det på en så lille plads ikke lade sig gøre at lave en komplet pascalcompiler. I BLS Pascal er der derfor givet afkald på nogle Pascal faciliteter, bl. a. brugerdefinerede variabeltyper, mængder og filtyper. Imidlertid er alle sætningskonstruktioner bibeholdt, ligesom procedurer og funktioner både kan tage value- og var-parametre. De grundlæggende datatyper INTEGER, REAL og BOOLEAN er også bibeholdt, mens datatypen CHAR er blevet erstattet af en den mere fleksible datatype STRING.

Denne manual definerer fuldstændigt BLS Pascal sproget, og bør derfor gennemlæses grundigt før programmeringen påbegyndes.

BLS Pascal systemet og de dertil hørende manualer er udviklet af Anders Hejlsberg i efteråret 1980.

1.1 SYMBOLER

Følgende tegn er de grundlæggende symboler i Pascal:

Bogstaver: A til Z, a til z, '_' og ' '.
 Talcifre: 0 1 2 3 4 5 6 7 8 9
 Symboler: + - * / = < > () [] . , ; : ' { }

Nogle operatorer og seperatorer er sammensat af to tegn:

1. <> <= >= := ..
2. (. og .) kan bruges i stedet for [og].
3. (* og *) kan bruges i stedet for { og }.

1.2 RESERVEREDE ORD OG STANDARDIDENTIFIERE

Reserverede ord kan ikke benyttes som brugerdefinerede identifiere. De reserverede ord er:

✓ AND	✓ FOR	✓ PROCEDURE
✓ ARRAY	✓ FUNCTION	✓ PROGRAM
✓ BEGIN	✓ GOTO	✓ REAL
✓ BOOLEAN	✓ IF	✓ REPEAT
✓ CASE	✓ INIT	✓ SHIFT
✓ DIV	✓ INTEGER	✓ STRING
✓ DO	✓ LABEL	✓ THEN
✓ DOWNTO	✓ MOD	✓ TO
✓ ELSE	✓ NOT	✓ UNTIL
✓ END	✓ OF	✓ VAR
✓ EXOR	✓ OR	✓ WHILE
✓ EXTERNAL	✓ OTHERS	

I BLS Pascal findes også et antal standardidentifiere, der ikke er reserverede, men som er navne på prædefinerede procedurer, funktioner og konstanter. Disse identifiere kan overskrives med brugerens egne definitioner.

✓ abs	✓ left	✓ read
✓ addr	✓ ln	✓ readln
✓ arctan	✓ load	✓ right
✓ call	✓ maxint	✓ round
✓ chr	✓ mem	✓ save
✓ concat	✓ mid	✓ sin
✓ cos	✓ odd	✓ sqr
✓ empty	✓ ord	✓ sqrt
✓ exp	✓ pi	✓ succ
✓ false	✓ plot	✓ true
✓ frac	✓ point	✓ trunc
✓ int	✓ pred	✓ write
✓ keyboard	✓ random	✓ writeln

Der skelnes ikke mellem små og store bogstaver i reserverede ord og standardidentifiere.

1.3 SEPERATORER

Blanktegn, linieskift og kommentarer anses for seperatorer. I-mellem to Pascal elementer skal der være mindst een seperator.

2.1 IDENTIFIERE

En identifier bruges til at navngive konstanter, procedurer, funktioner, variable og labels. En identifier består af et bogstav efterfulgt af et vilkårligt antal bogstaver, talcifre eller '.'-tegn. Eksempler:

PASCAL Pascal NAVN.41.CODE

Compileren skelner mellem små og store bogstaver i brugerdefinerede identificere.

2.2 TAL

Tal skrives i decimal eller hexadecimal notation. Hexadecimal notation vælges ved at skrive et \$-tegn foran tallet. Bogstavet E foran skalafaktoren udtales som 'gange 10 opløftet i'.
Eksempler:

1 100 \$25EC 0.138 5E10 87.13E-8

Talkonstanter må ikke indeholde blanktegn.

2.3 STRENGE

En streng er en sekvens af tegn omsluttet af enkelte anførelsestegn. Anførelsestegnet kan indgå i strengen, hvis det skrives dobbelt. Eksempler:

'BLS Pascal' 'A' 'A' ' 'That''s all folks'

2.4 KOMMENTARER

En kommentar er en sekvens af tegn, der kan fjernes fra programmet, uden at dettes mening ændres. Kommentarer skal omslutes af symbolerne { og } eller af symbolerne (* og *).
Eksempel:

(* Dette er en kommentar *)

En variabels datatype er afgørende for hvilke værdier, den kan antage i programmet. I BLS Pascal er der fire datatyper: Integer, real, boolean og string.

3.1 INTEGERS

En integer er et heltal i området $-32768 \leq I \leq 32767$. Overflow på integers detekteres ikke.

3.2 REALS

En real er et reelt tal i områderne:

$$-1.7014118346E38 \leq R \leq -2.9387358770E-39$$
$$R = 0$$
$$2.9387358770E-39 \leq R \leq 1.7014118346E38$$

I BLS Pascal regnes reelle tal med 11+ betydende cifre. Ved underflow på reelle tal sættes resultatet lig 0. Ved overflow stopper programmet med en fejlmeddelelse.

3.3 BOOLEANS

I BLS Pascal opfattes Integers og Booleans ens. Typen boolean er kun medtaget for at give kompatibilitet med andre Pascal-compile. Booleans bør ikke antage andre værdier end true (-1) eller false (0).

3.4 STRINGS

Når en string erklæres, angiver man den maksimale længde, denne kan antage (mellem 1 og 255). Eksempler:

```
STRING (.32.)
```

```
STRING (.stringsize.)
```

3.5 ARRAYSTRUKTURER

Et array består af et antal dataelementer, der alle er af samme type. Et element refereres ved at angive et (eller flere) index, der afgør, hvor i strukturen elementet findes. Når en arraystruktur erklæres, angiver man den nedre og øvre grænse for hver enkelt dimension. Eksempler:

```
ARRAY (.1..10.) OF INTEGER
```

```
ARRAY (.0..maxsize.) OF STRING (.32.)
```

```
ARRAY (.-5..11,29..45.) OF REAL
```

Variable i en n-dimensional arraystruktur refereres ved at angive arraystrukturens identifier efterfulgt af n heltalsudtryk, adskilt af kommaer, omsluttet af firkantparenteser. Eksempler:

```
data(.12.)
```

```
b(.i+j,7.)
```

```
navne(.pil(.8.),3.)
```


3.5.1 mem arraystrukturen

mem arraystrukturen er et prædefineret endimensionalt array, der repræsenterer computerens memory. Hver enkelt element svarer til en byte hvis adresse er dens index i arraystrukturen. Et element kan således kun antage værdier mellem 0 og 255. Tilskrives en værdi større end 255, bliver den aktuelle værdi tallet modulus 256. Eksempler:

(1) `i:=mem(.$C00.) AND $16;`

se evt side 13

(2) `FOR p:=1 TO length(s) DO
mem(.offset+p.):=ord(mid(s,p,1));`

(3) *indholdet af 0C00H AND med 16H.*

4: ERKLÆRINGER OG DEFINERINGER

Et program består af tre dele:

1. Programhovedet
2. Erklæringsdelen
3. Sætningsdelen

I programhovedet erklæres programmets navn og input/output parametre. Eksempler:

```
PROGRAM konverter;
```

```
PROGRAM beregning(input,output);
```

I BLS Pascal er programhovedet ikke nødvendigt, og hvis det er tilstede opfattes alt, mellem symbolet PROGRAM og det næste semikolon, som en kommentar.

Erklæringer skal foretages i rækkefølgen:

1. Labelerklæringer
2. Konstantdefineringer
3. Variabelerklæringer
4. Procedure/Funktion erklæringer

Ingen af de ovennævnte punkter behøver at være til stede (således kan erklæringsdelen godt være tom).

4.1 LABELERKLÆRINGER

I labelerklæringsdelen, der starter med det reserverede ord LABEL, skrives identifikatorerne for alle de labels der bruges i programmet adskilt af kommaer. I BLS Pascal kan en label enten være et fortegnsfrit heltal eller en identifikator. Eksempel:

```
LABEL 1,error,999,stop;
```

En label defineres ved at skrive navnet efterfulgt af et kolon. Eksempel:

```
999: write('Slut');
```

En label bør kun refereres indenfor den blok hvori den er erklæret.

4.2 KONSTANT DEFINERINGER

Konstantdefineringsdelen startes med det reserverede ord CONST. Navnet på en konstant er en identifikator. Konstante værdier kan være tal eller strenge. Eksempel:

```
CONST
  antal=45;
  max=193.158;
  min=-max;
  navn='Jensen';
```


Følgende standardkonstanter behøver ingen definering:

pi	Real	3.1415926536.
true	Boolean	Sand (-1).
false	Boolean	Falsk (0).
maxint	Integer	32767.
empty	String	' ' (Den tomme streng).

4.3 VARIABELERKLÆRINGER

Alle variable, der anvendes i et program, skal erklæres i en VAR-erklæringsdel. Denne definerer, hvilken datatype variabelen skal være, og dermed hvilke værdier den kan antage. Eksempel:

```
VAR
  i,j,k: INTEGER;
  x.coor,y.coor: REAL;
  navne: ARRAY (.1..100.) OF STRING (.32.)
```

Flere variable af samme datatype kan erklæres samtidigt ved at skrive deres identificere adskilt af kommaer. En variabel kan kun bruges i den blok, hvori den erklæres, samt de blokke, der erklæres inden for denne blok.

Under indgangen til en blok bliver alle de heri erklærede variable nulstillet, dvs. reals og integers antager værdien 0, booleans antager værdien false og strings antager værdien empty.

4.4 PROCEDURE- OG FUNKTIONSERKLÆRINGER

Procedureerklæringsdelen bruges til at definere underprogrammer i hovedprogrammet (se afsnit 7).

Funktionserklæringsdelen bruges til at definere underprogrammer der returnerer en værdi (se afsnit 8).

I BLS Pascal er der 4 klasser af operatører. Operatoren NOT har den højeste prioritet, efterfulgt af de såkaldte multiplicerende operatører (* / DIV MOD AND SHIFT), igen efterfulgt af de såkaldte adderende operatører (+ - OR EXOR), og til sidst sammenligningsoperatørerne (= <> > < >= <=).

Alle operatører der tillader integers tillader også booleans.

5.1 NOT OPERATOREN

Operatoren NOT indikerer, at operanden skal komplementeres. Operanden skal være af typen boolean eller integer. Eksempler:

```
NOT true      = false
NOT false     = true
NOT 5         = -6
```

5.2 MULTIPLICERENDE OPERATORER

Operator	Operation	Operandtype	Resultattype
*	Multiplikation	real, integer	real, integer
/	Division	real, integer	real
DIV	Heltalsdivision	integer	integer
MOD	Modulus	integer	integer
SHIFT	Logisk skiftning	integer	integer
AND	Logisk AND	integer	integer

Operationen I SHIFT J har følgende virkning: I skiftes J gange mod venstre, hvis J er positiv og -J gange mod højre, hvis J er negativ. Hvis ABS(J) er større end 15 er resultatet altid nul.

5.3 ADDERENDE OPERATORER

Operator	Operation	Operandtype	Resultattype
+	Addition	real, integer	real, integer
-	Subtraktion	real, integer	real, integer
OR	Logisk INCLUSIVE OR	integer	integer
EXOR	Logisk EXCLUSIVE OR	integer	integer

Hvis + eller - bruges med kun een operand, indikerer de henholdsvis identisk og inverteret fortegn.

5.4 SAMMENLIGNENDE OPERATORER

De sammenlignende operatører = , <> , > , < , >= , <= betyder henholdsvis lig med, forskellig fra, større end, mindre end, større end eller lig med og mindre end eller lig med. Alle datatyper kan indgå som operander, dog må tal ikke sammenlignes med strings.

Resultatet af en sammenligning har enten værdien true (-1) eller false (0), og typen integer (boolean).

Ved sammenligning af strings, sammenlignes ASCII værdien af hver enkelt karakter, til en forskel mødes, eller en af de to strenge ikke er længere. Eksempler:

' '<'A'
'HIJK'<'HIJKLMN'

true
true

-11-

5.5 PARANTESER I UDTRYK

Udtrykket

$a+b*c$

beregnes ved at multiplicere b og c og addere a hertil, idet multiplikation har højere prioritet end addition.

Ønsker man at ændre denne rækkefølge kan der indsættes paranteser i udtrykket, idet paranteser altid har højeste prioritet:

$(a+b)*c$

Nu bliver additionen udført før multiplikationen.

5.6 FUNKTIONSKALD

Et funktionskald (til enten en standardfunktion eller en brugerdefineret funktion) kan indgå i et udtryk ved at angive funktionens identifier eventuelt efterfulgt af en parameterliste. Funktionens resultat kan betragtes som en variabel af samme datatype som funktionen. Eksempler:

```
sin(y)+cos(x)
concat('Navn: ',fornavn,' ',efternavn)
arctan(1.0)*4.0
afstand(x,y)>5 AND (z=0)
```


6: SÆTNINGER

Sætninger i Pascal kan opdeles i to hovedgrupper: Simple sætninger, der ikke kan indeholde andre sætninger, og strukturerede sætninger, der indeholder andre sætninger.

Enhver sætning kan forudgås af en label, der kan refereres i en GOTO sætning (se afs. 4.1 og 6.1.3).

6.1 SIMPLE SÆTNINGER

I BLS Pascal er der fem simple sætninger: Tilskrivningssætningen, proceduresætningen, GOTO-sætningen, INIT-sætningen og den tomme sætning.

6.1.1 Tilskrivningssætningen

Målet med en tilskrivningssætning er at tilskrive en variabel eller et funktionsresultat en værdi.

I en tilskrivning bruges operatoren `:=` til at adskille variabelen (eller funktionen) og udtrykket.

Variablen (eller funktionen) og udtrykket skal være af samme datatype, dog med den undtagelse at reals godt kan tilskrives integer (heltallige) værdier. Eksempler:

```
x:=y+z           (* x antager værdien y plus z *)
```

Hvis en strengvariabel tilskrives et strengudtryk, der er længere end strengvariablens maksimale længde, overføres kun de venstremest justerede tegn.

6.1.2 Proceduresætningen

En proceduresætning bruges til at aktivere den procedure, hvis identifier indgår i sætningen. Proceduresætningen kan desuden indeholde en liste over aktuelle parametre, der under kaldet bliver substitueret af de tilsvarende formelle parametre. Eksempler:

```
sorter(navne);
ombyt(x,y);
plot(x,round(sin(x*f)*20)+20,1);
```

6.1.3 GOTO-sætningen

En GOTO-sætning indikerer, at programudførslen skal fortsættes fra den label der refereres.

Ved brugen af GOTO-sætninger bør følgende regler iagttages:

- 1) En label bør kun refereres inden for den blok, hvori den er erklæret. Det er således ikke tilladt at hoppe ind og ud af procedureblokke.
- 2) Et hop ud af og ind i en FOR-sætning er ikke tilladt.

6.1.4 INIT-sætningen

En INIT-sætning bruges til at tilskrive en arraystruktur konstante værdier. De givne værdier skal være af samme datatype som arraystrukturen.

Hvis der er færre elementer i INIT-listen end i arraystrukturen, bliver kun de første elementer initieret. Eksempel:

```
VAR
  data: ARRAY (.1..6.) OF INTEGER;
BEGIN
  data(.6.)=100;
  INIT data TO -15,6,99,341;
  :
  :
END.
```

Efter INIT-sætningen har elementerne i DATA de følgende værdier:

data(.1.)=-15	data(.2.)=6	data(.3.)=99
data(.4.)=341	data(.5.)=0	data(.6.)=100

Hvis arraystrukturen har flere dimensioner bliver der talt bagfra. Eksempel:

```
VAR
  a: ARRAY (.1..3,1..3.) OF INTEGER;
BEGIN
  INIT a TO -9,6,-3,18,41,38,1357,-1,5;
  :
  :
END.
```

Svarer til:

```
VAR
  a: ARRAY (.1..3,1..3.) OF INTEGER;
BEGIN
  a(.1,1.):=-9;   a(.1,2.):=6;   a(.1,3.):=-3;
  a(.2,1.):=18;   a(.2,2.):=41;  a(.2,3.):=38;
  a(.3,1.):=1357; a(.3,2.):=-1;  a(.3,3.):=5;
  :
  :
END.
```

INIT-sætningen kan også bruges til at initiere en blok af memory. Eksempel:

```
INIT mem(.mc.) TO $EF,$41,$42,$43,$00,$C9;
```

Hvis heltalsvariablen mc har værdien \$D00 gælder der efter endt udførsel af denne sætning at adressen \$D00 har værdien \$EF, adressen \$D01 værdien \$41 etc.

6.1.5 Tomme sætninger

Den tomme sætning indeholder ingen symboler (undtaget kommentarer) og giver således heller ingen aktion. En tom sætning forekommer de steder hvor Pascalsyntaxen forventer en sætning, men hvor der ingen er. Eksempler:

```
BEGIN END;
WHILE digit AND (a>17) DO (* ingenting *);
REPEAT (* vent *) UNTIL keyboard;
```


6.2 STRUKTUREREDE SÆTNINGER

En struktureret sætning er en sætning der blandt andet indeholder andre sætninger.

6.2.1 Sammensat sætning

I nogle tilfælde må man kun anvende en enkelt sætning. Såfremt man i denne situation ønsker at skrive flere sætninger, skal de omsluttet af BEGIN og END, der virker som sætningsparanteser. Eksempel:

```
IF a=1 THEN
  BEGIN
    z:=x; x:=y; y:=z  (* ombyt x og y ved hjælp af z *)
  END;
```

De enkelte sætninger, i den sammensatte sætning, adskilles med semikolon. Der behøver ikke at være et semikolon efter den sidste sætning i den sammensatte sætning.

6.2.2 Betingede sætninger

En betinget sætning er en sætning, der på grundlag af et valg udfører en af de indeholdte sætninger.

6.2.2.1 IF-sætningen

IF-sætningen bruges i de tilfælde, hvor en sætning kun skal udføres, hvis et (boolean) udtryk er sandt. Hvis udtrykket er falsk, gælder der enten, at der ikke udføres nogen aktion, eller at sætningen efter ELSE symbolet skal udføres. Sætningen:

```
IF <e1> THEN IF <e2> THEN <s1> ELSE <s2>;
```

skal forstås på følgende måde:

```
Hvis <e1> er falsk, udføres der ingen aktion.
Hvis <e1> er sand, og <e2> er sand, udføres <s1>.
Hvis <e1> er sand, og <e2> er falsk, udføres <s2>.
```

Generelt gælder der, at en ELSE-del hører sammen med den sidste IF-del, der mangler en ELSE-del.

Eksempler:

```
IF x<1.5 THEN z:=x+y ELSE z:=1.5;
IF navn=empty THEN navn:='Ikke opgivet';
```

6.2.2.2 CASE-sætningen

CASE-sætningen består af et udtryk (kaldet valgudtrykket) og en liste af sætninger hver foregået af en værdiliste bestående af en eller flere konstanter af samme type som valgudtrykket. En sætning bliver udført, hvis værdien af valgudtrykket findes i sætningens værdiliste. Hvis ingen af værdilisterne indeholder en konstant, svarende til valgudtrykkets værdi, bliver sætningen, der er forudgået af OTHERS:, udført. Hvis værdilisten OTHERS ikke er tilstede, udføres der ingen aktion i det sidstnævnte tilfælde. Værdilisten OTHERS: skal, hvis den er tilstede, være den sidste værdiliste. Valgudtrykket skal være af typen integer, boolean eller string (reals er ikke tilladt).

Eksempler:

```
CASE operator OF
```

```
  '+': x:=x+y;
```

```
  '-': x:=x-y;
```

```
  '*': x:=x*y;
```

```
  '/': x:=x/y
```

```
END;
```

```
CASE i OF
```

```
  1: write('en');
```

```
  2: write('to');
```

```
  3,4,5: write('flere');
```

```
  OTHERS: write('mange')
```

```
END;
```

Det er ikke nødvendigt med et semikolon efter den sidste sætning i CASE-sætningen (men det må godt være tilstede).

6.2.3 Repetitionssætninger

En repetitionssætning bruges de steder, hvor man ønsker at gentage en, eller flere, sætninger. Er antallet af gentagelser på forhånd kendt, bør man bruge FOR-sætningen; ellers kan WHILE- og REPEAT-sætningerne bruges.

6.2.3.1 WHILE-sætningen

Udtrykket, der kontrollerer WHILE-sætningen, skal være af typen boolean (integer). Så længe WHILE-udtrykket er sandt, gentages sætningen efter DO. Eksempel:

```
WHILE a<1000 do
  BEGIN
    a:=sqr(a); b:=b+1
  END;
```

6.2.3.2 REPEAT-sætningen

Udtrykket, der kontrollerer REPEAT-sætningen, skal være af typen boolean (integer). Hvergang sætningerne mellem REPEAT og UNTIL er udført, testes sandhedsudtrykket. Hvis dette er falskt udføres sætningerne igen. Eksempel:

```
REPEAT
  read(digit); write(digit);
  number:=number*10+ord(digit)-48;
UNTIL number>10000;
```

6.2.3.3 FOR-sætningen

FOR-sætningen indikerer, at en sætning skal udføres gentagne gange, mens en stigende eller faldende række af værdier bliver tilskrevet en variabel, kaldet styrevariablen. Styrevariablen, startværdien og slutværdien skal være af typen integer. Værdierne kan enten stige i spring af 1 (TO) eller falde i spring af 1 af (DOWNTO).

Hvis slutværdien er mindre end startværdien ved TO, eller slutværdien er større end startværdien ved DOWNTO, udføres sætningen i FOR-løkken ikke.

Det er ikke tilladt at afbryde en FOR-sætning med en GOTO-sætning. I stedet bør en WHILE- eller REPEAT-sætning bruges.

Eksempler:

```
FOR i:=1 TO 100 DO write(i:3,sqr(i):6);
```

```
FOR i:=1 TO 100 DO FOR j:=1 TO 10 DO  
BEGIN  
  IF a(.i,j.)>5 THEN a(.i,j.):=5;  
  c:=c+a(.i,j)  
END;
```

Efter endt udførsel af en FOR-sætning, gælder der følgende om styrevariablens værdi:

Hvis FOR-sætningen blev oversprunget, er styrevariablen lig startværdien.

Hvis FOR-sætningen blev aktiveret, er styrevariablen lig slutværdien plus eller minus 1, alt efter om der blev brugt TO eller DOWNT0 i FOR-sætningen.

7: PROCEDURER

En procedure (underprogram) er en selvstændig enhed i det samlede program, med egne labels, konstanter, variable og eventuelt andre procedurer og funktioner. En procedure aktiveres fra en proceduresætning (se afsnit 6.1.2).

7.1 PROCEDUREERKLÆRINGER

Procedureerklæringer består generelt af 3 dele:

- 1) Procedurehovedet
- 2) Erklæringsdelen
- 3) Sætningsdelen

Eksempel:

```
PROCEDURE ombyt(VAR x,y: REAL);
VAR
  temp: REAL;  (* hjælpevariabel *)
BEGIN
  temp:=x; x:=y; y:=temp  (* ombyt x og y *)
END;
```

7.1.1 Procedurehovedet

I procedurehovedet erklæres procedures navn og dens eventuelle parametre. Procedures navn er en identifier og skal følge de regler, der er givet for sådanne. Procedurehovedet kan eventuelt efterfølges af symbolet EXTERNAL. Dette betyder at proceduren er et underprogram i maskinkode, der starter i den adresse, der er givet ved heltalskonstanten, der efterfølger EXTERNAL. Hvis en procedure er erklæret som EXTERNAL, er erklæringsdelen og sætningsdelen tomme. Eksterne underprogrammer er yderligere beskrevet i APPENDIX E.

7.1.2 Erklæringsdelen

En procedures erklæringsdel har samme form som et programs (se afsnit 4).

7.1.3 Sætningsdelen

En procedures sætningsdel har samme form som et programs (se afsnit 6), dog med den undtagelse, at man i tilskrivninger og udtryk, ud over de lokale variable, også kan bruge procedures parametre.

7.2 STANDARDPROCEDURER

Standardprocedurerne i BLS Pascal behøver ingen erklæring, og kan overskrives med brugerens egne erklæringer.

call(a) Genererer et kald til den adresse, der er givet ved heltalsudtrykket a.

screen(x,y) Flytter cursoren til linie y position x; x og y er heltalsudtryk. Hvis en af koordinatværdierne er ulovlige, bliver den nuværende værdi i denne koordinat bibeholdt. screen proceduren kan således bruges som tabulator ved at sætte y-koordinaten lig 0.

`plot(x,y,f)`

Udfører en operation på det semigrafiske punkt x,y (hvor x og y er heltalsudtryk i områderne $0 \leq x \leq 95$ og $0 \leq y \leq 47$), afhængigt af heltalsudtrykket f 's værdi:

$f=0$: Slukkes
 $f=1$: Tændes
 $f=2$: Inverteres

I proceduren `plot` er der kompenseret for skærmadresseringen, således at punkter med y -koordinater mellem 0 og 2 bliver plottet på skærmens øverste linie.

Standardprocedurerne for input og output er beskrevet i afsnit 10.

8: FUNKTIONER

En funktion er et underprogram der beregner, og returnerer, en værdi. En funktion aktiveres ved at lade den indgå i et udtryk (se afsnit 5.6).

8.1 FUNKTIONSERKLÆRINGER

En funktionserklæring består af tre dele:

- 1) Funktionshovedet
- 2) Erklæringsdelen
- 3) Sætningsdelen

Eksempler:

```
FUNCTION afstand(x,y: REAL): REAL;
BEGIN
  afstand:=sqrt(sqr(x)+sqr(y))
END;
```

```
FUNCTION snit(VAR a: ARRAY(.1..100.) OF INTEGER;
              start,slut: INTEGER): REAL;
VAR
  i,s: INTEGER;
BEGIN
  FOR i:=start TO slut DO
    s:=s+a(.i.);
  snit:=s/(slut-start+1)
END;
```

8.1.1 Funktionshovedet

I funktionshovedet defineres funktionens navn og resultattype og dens eventuelle parametre. Funktionens navn er en identifier og skal følge de regler, der er givet for sådanne. En funktion kan kun returnere værdier af standard datatyper, dvs. real, integer, boolean eller string. Funktionens parametre skal følge de regler, der er givet i afsnit 9. Funktionshovedet kan eventuelt efterfølges af symbolet EXTERNAL. Dette betyder, at funktionen er et underprogram i maskinkode, der starter i den adresse, der er givet ved heltalskonstanten, der efterfølger EXTERNAL. Hvis en funktion er erklæret som EXTERNAL, er erklæringsdelen og sætningsdelen tomme. Eksterne underprogrammer er yderligere beksrevet i APPENDIX E.

8.1.2 Erklæringsdelen

En funktions erklæringsdel har samme form som et programs (se afsnit 4).

8.1.3 Sætningsdelen

En funktionss sætningsdel har samme form som et programs (se afs. 6), dog med den undtagelse, at man i tilskrivninger og udtryk, ud over de lokale variable, også kan bruge funktionens parametre. Desuden skal man på et tidspunkt tildele funktionens identifier en værdi, der afgør funktionens resultat.

8.2 STANDARDFUNKTIONER

Standardfunktionerne i BLS Pascal behøver ingen erklæring, og kan overskrives med brugerens egne erklæringer.

8.2.1 Reelle funktioner

De reelle standardfunktioner tager et argument af typen real og giver et resultat af typen real.

abs(x)	Den absolutte værdi af x.
sqr(x)	Kvadratet af x.
sqrt(x)	Kvadratroden af x.
sin(x)	Sinus til x i radianer.
cos(x)	Cosinus til x i radianer.
arctan(x)	Arccus tangens til x i radianer.
ln(x)	Den naturlige logaritme af x.
exp(x)	Exponentialfunktionen: 2.7182818285 opløftet i potens x.
int(x)	Heltalsdelen af x. Returnerer det største heltal, der er mindre end x, hvis $x \geq 0$, eller det mindste heltal, der er større end x, hvis $x < 0$.
frac(x)	Decimaldelen af x med samme fortegn som x (er defineret som $\text{frac}(x) = x - \text{int}(x)$).

8.2.2 Heltalsfunktioner

Heltalsfunktionerne tager et argument af typen integer og giver et resultat af typen integer.

abs(i)	Den absolutte værdi af i.
sqr(i)	Returnerer $i*i$.
succ(i)	Returnerer $i+1$.
pred(i)	Returnerer $i-1$.
odd(i)	Returnerer værdien false (0), hvis i er lige og true (-1), hvis i er ulige.

8.2.3 Stringfunktioner

length(s)	Returnerer længden af strengen s. Resultatet er af typen integer.
-----------	---

mid(s,p,n)	Returnerer den streng, der fremkommer ved at tage n tegn fra strengen s startende fra position p. n og p skal være heltalsudtryk.
------------	---

*Q1 = LARS JUUL
Q1 = MID(Q1,1,4) = LARS*

mid(s,p)	Returnerer alle tegn til venstre ^{HOJRE} for position p i strengen s. p skal være et heltalsudtryk.
----------	---

Q1 = MID(Q1,2)

LARS JUUL

`left(s,n)` Returnerer de venstre n tegn fra strengen s. n skal være et heltalsudtryk.

`right(s,n)` Returnerer de højre n tegn fra strengen s. n skal være et heltalsudtryk.

`concat(s1, s2,...,sn)` Returnerer string 1, string 2 frem til string n sammensat i den nævnte rækkefølge.

8.2.4 Konverteringsfunktioner

`trunc(x)` x er et udtryk af typen real. Resultatet er den største integer, der er mindre end eller lig x, hvis $x \geq 0$, eller den mindste integer, der, er større end eller lig x, hvis $x < 0$.

trunc(2.8) = 2

`round(x)` x er et udtryk af typen real. Resultatet, der er af typen integer, er den afrundede værdi af x, hvilket svarer til:

round(2.8) = 3

$$\text{round}(x) = \begin{cases} \text{trunc}(x+0.5), & \text{for } x \geq 0 \\ \text{trunc}(x-0.5), & \text{for } x < 0 \end{cases}$$

`chr(i)` i er et heltalsudtryk. Resultatet er en streng, med længden 1, hvis tegn har ASCII-værdien i.

`ord(s)` s er et strengudtryk. Resultatet er en integer, der svarer til ASCII-værdien af det første tegn i s. Hvis s er den tomme streng returneres 0.

8.2.5 Andre standardfunktioner

`addr(v)` Returnerer variablen v's memoryadresse. En arraystrukturens adresse kan fås ved at referere til det første element i strukturen.

`random` Returnerer et tilfældigt tal af typen real i området $0 \leq r < 1$.

`random(i)` i er et heltalsudtryk. Returnerer et tilfældigt heltal i området $0 \leq t < i$.

`keyboard` Scanner tastaturet, og returnerer den nedtrykkede tasts ASCII værdi. Hvis ingen taster er nedtrykket returneres et 0.

`point(x,y)` Returnerer true (-1), hvis det semigrafiske punkt (x,y) er tændt og false, hvis det er slukket.

Parametre bruges til at overføre værdier til en procedure eller en funktion. I selve underprogrammet kan parametrene betragtes som almindelige variable.

9.1 PARAMETERTYPER

I BLS Pascal er der to parametertyper: Value-parametre og var-parametre. Parametre specificeret med VAR i procedurehovedet eller funktionshovedet kaldes var-parametre.

9.1.1 Value-parametre

Hvis en parameter er en value-parameter, oprettes der ved procedurekaldet en variabel (den formelle parameter), hvis startværdi er givet den aktuelle parameter.

Ved enkeltvariable skal den aktuelle parameter være et udtryk (hvoraf en variabel er et tilfælde). Ved arraystrukturer skal den aktuelle parameter være en arraystruktur af samme type, og med samme antal elementer, som den formelle parameter.

Tilskrivninger til den formelle parameter berører ikke den aktuelle parameter.

9.1.2 Var-parametre

Hvis en parameter er en var-parameter, er både den aktuelle og den formelle parameter lig den samme lagerplads. Under udførelsen af underprogrammet repræsenterer den formelle parameter den aktuelle parameter, og tilskrivninger til den formelle parameter vil ændre den aktuelle parameter tilsvarende. Aktuelle var-parametre skal altid være en variabels eller en arraystrukturens identifier.

9.2 FORMELLE OG AKTUELLE PARAMETRE

Parametre, der nævnes i procedure/funktions-erklæringen, kaldes formelle parametre. Parametre der nævnes i procedure/funktionskaldet kaldes aktuelle parametre. De formelle og aktuelle parametre skal stemme overens med hensyn til antal, rækkefølge og type, dog med følgende undtagelser:

- 1) Hvis en formel parameter er en value-parameter af typen real, må den aktuelle parameter godt være af typen integer. Dette gælder imidlertid ikke for var-parametre.
- 2) Hvis en formel parameter er en value-parameter af typen string, må den aktuelle stringparameter have hvilkensomhelst længde. Dog gælder der, at hvis den aktuelle stringparameters længde er større end den formelle stringparameters maksimale længde, bliver kun de venstremest justerede tegn overført. Ovennævnte gælder ikke for var-parametre.
- 3) Hvis den formelle parameter er en arraystruktur, skal den aktuelle parameter være en arraystruktur af samme type og med samme antal elementer. Imidlertid behøver der ikke at være overensstemmelse mellem øvre og nedre grænser og antallet af dimensioner.

10: INDLÆSNING OG UDLÆSNING

I Pascal varetages indlæsning fra og udlæsning til konsollen af fire standardprocedurer (read, readln, write og writeln). BLS Pascal indeholder yderligere to standardprocedurer til indlæsning og udlæsning af arraystrukturer fra og til bånd-optager (load og save).

10.1 INDLÆSNING FRA KONSOLLEN

Indlæsning fra konsollen varetages af standardprocedurerne read og readln.

10.1.1 read proceduren

READ proceduren kan indlæse enkeltvariable af alle datatyper. Formatet af READ proceduresætningen er:

```
read(v1,v2,...,vn);
```

Hvilket svarer til:

```
read(v1); read(v2); ... read(vn);
```

Under indtastningen må brugeren benytte følgende specialnøgler:

<BS>	Sletter det sidst indtastede tegn.
<ESC>	Sletter hele indtastningen.
<ENTER>	Afslutter indtastningen.

Ved indlæsning af talvariable (reals og integers) gælder der, at tallet må foregås af et vilkårligt antal blanktegn, men det må ikke indeholde blanktegn. Det skrevne tal skal følge de samme regler for formater som talkonstanter i et program, og skal være af samme datatype som den variabel det indlæses i (dog med den undtagelse at heltal godt må indlæses i reals). En indlæsning skal afsluttes med et linieskift. Linieskiftet bliver ikke udskrevet. Hvis det indtastede tal indeholder fejl (overflow eller ulovlige tegn), bliver indtastningen slettet og, cursoren returnerer til startpositionen, hvorefter en ny indtastning kan foretages.

Ved indtastning af strengvariable bliver alle de indlæste tegn gemt i strengen. Der kan maksimalt indlæses det antal tegn der, svarer til strengvariablens længde, eller hvis strengvariablens maksimale længde er større end 64, 64 tegn. Hvis strengvariablens maksimale længde er 1, fortsætter programkørslen umiddelbart efter en tast er nedtrykket, uden at tegnet bliver udskrevet.

10.1.2 readln proceduren

Forskellen på read proceduren og readln proceduren er, at det afsluttende linieskift bliver udskrevet med readln. Formatet af readln proceduresætningen er:

```
readln(v1,v2,...,vn);
```

Hvilket svarer til:

```
readln(v1); readln(v2); ... readln(vn);
```


10.2 UDLÆSNING TIL KONSOLLEN

Udlæsning til konsollen varetages af standardprocedurerne `write` og `writeln`.

10.2.1 write proceduren

WRITE proceduren kan udskrive enkeltvariable af alle datatyper. Formatet af WRITE proceduresætningen er:

```
write(p1,p2,...,pn);
```

Hvilket svarer til:

```
write(p1); write(p2); ... write(pn);
```

hvor $p_1, p_2, \dots, p_n$ er såkaldte write-parametre. Write-parametrene kan have forskellig form alt efter hvilken datatype, der skal udskrives. I beskrivelsen er m og n synonymer for heltalsudtryk.

INTEGER:

- i Udtrykket i udskrives i frit format.
- $i:n$ Udtrykket i udskrives højrejusteret i et felt på n tegn.

REAL:

- r Udtrykket r udskrives i exponentiel notation i et felt på 18 tegn. Formatet af udskriften er:

```
" sd.dddddddddddEtdd"
```

hvor s enten er " " eller "-", d er et ciffer, og t er enten "+" eller "-".

- $r:n$ Udtrykket r udskrives i exponentiel notation. Formatet er afhængigt af n 's værdi:

```
n<8: "-d.dEtdd" eller "d.dEtdd"
```

```
8<=n<17: "sd.<digits>Etdd", hvor <digits> er n-6 cifre.
```

```
n>17: "<spaces>sd.dddddddddddEtdd", hvor <spaces> er n-17 blanktegn.
```

- $r:n:m$ Udtrykket r udskrives højrejusteret i fastkommanotation i et felt på n tegn med m decimaler. m skal være i området $0 \leq m \leq 24$. Hvis dette ikke er tilfældet, vælges exponentiel notation.

STRING:

- s Stringudtrykket s udskrives.
- $s:n$ Stringudtrykket s udskrives i et felt på n tegn. n er en heltalskonstant.

Generelt gælder der følgende for udskrifter:

- 1) Hvis n er mindre end længden på det der skal udskrives udvides feltet.
- 2) Hvis n er mindre end 0, betragtes n som værende 0.
- 3) Hvis n er større end 255, betragtes n som værende 255.

10.2.2 writeln proceduren

Forskellen på writeln og write procedurerne er, at der efter endt udskrift udskrives et linieskift, hvis writeln proceduren benyttes. Formatet for writeln proceduresætningen er:

```
writeln(p1,p2,...,pn);
```

Hvilket svarer til:

```
write(p1); write(p2); ... writeln(pn);
```

Hvis writeln proceduren kaldes uden parametre er resultatet, at der udføres et linieskift.

10.3 INDLÆSNING OG UDLÆSNING AF ARRAYSTRUKTURER

Indlæsning og udlæsning af arraystrukturer fra og til båndoptager varetages af standardprocedurerne load og save.

10.3.1 save proceduren

save standardproceduren kan udlæse arraystrukturer af alle datatyper til båndoptager. Formatet af save proceduresætningen er:

```
save(a);
```

hvor a er en arraystrukturens identifier. Ved procedurekaldet tændes tape-lysdioden, arraystrukturen udskrives, og lysdioden slukkes igen.

10.3.2 load proceduren

load standardproceduren kan indlæse arraystrukturer, der er udlæst med save proceduren. Formatet af load proceduresætningen er:

```
load(a,i);
```

Hvor a er en arraystrukturens identifier, og i er en heltalsvariabel, hvori en eventuel fejlkode kan returneres.

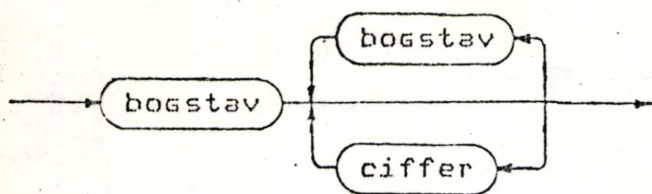
Ved procedurekaldet tændes tape-lysdioden. Denne slukkes igen når proceduren forlades, hvilket kan ske på en af de følgende måder:

Brugeren afbrød proceduren ved at trykke <ESC>.
En fejl forekom under indlæsningen.
Indlæsningen forløb OK.

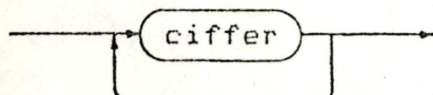
I heltalsvariablen i kan følgende værdier forekomme efter procedurekaldet:

- i=0: Indlæsningen forløb OK.
- i=1: Arraystrukturernes elementantal eller datatype stemmer ikke overens.
- i=2: Checksumfejl.
- i=3: Brugeren afbrød proceduren ved at trykke <ESC>.

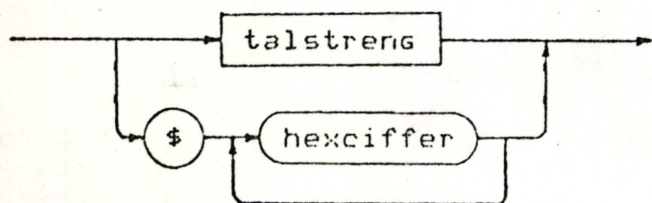
entifier (ident)



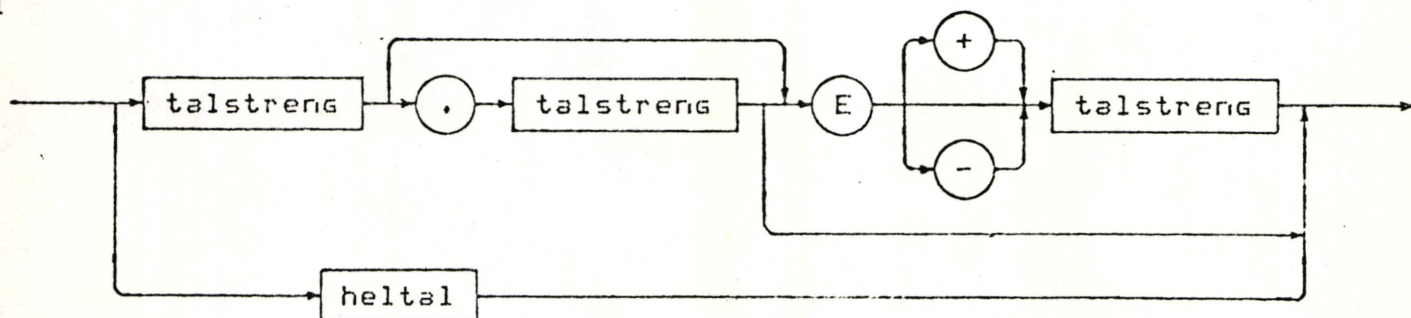
talstreng



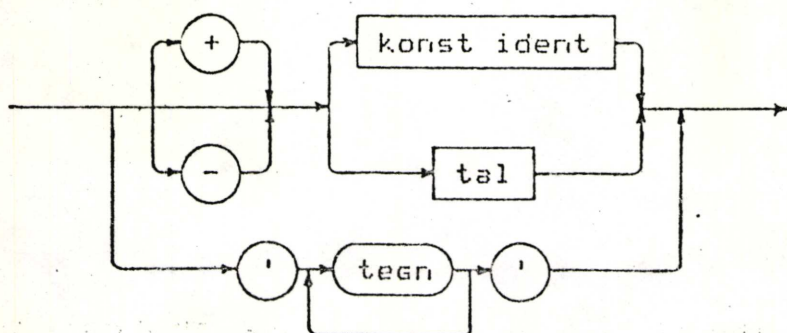
heltal



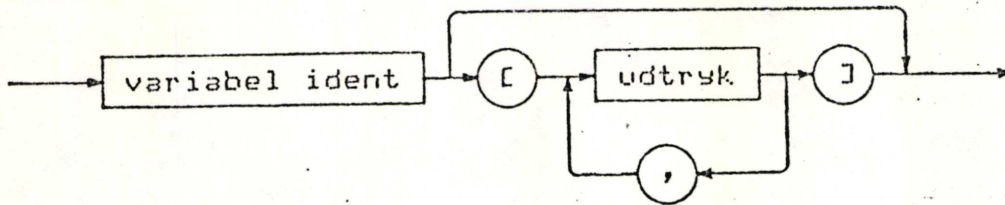
tal



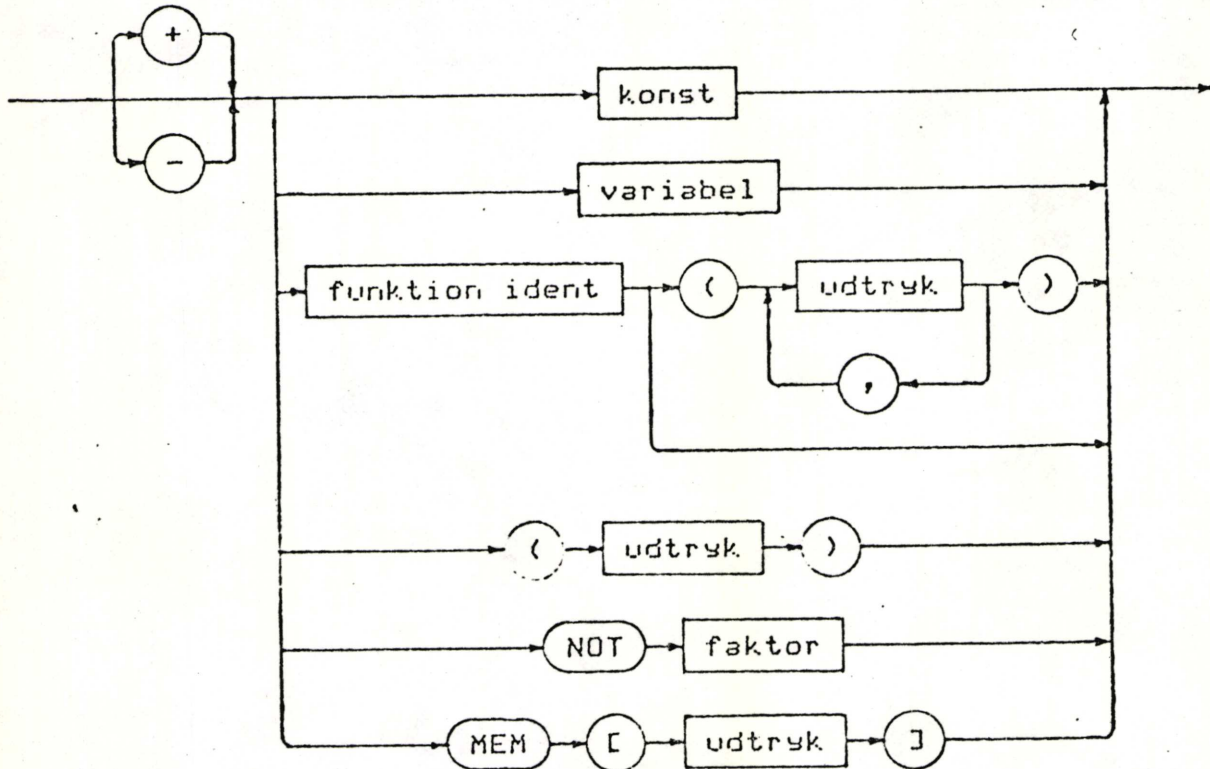
konstant (konst)



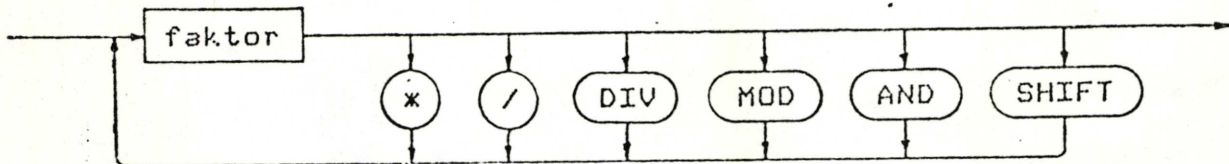
v. iabel



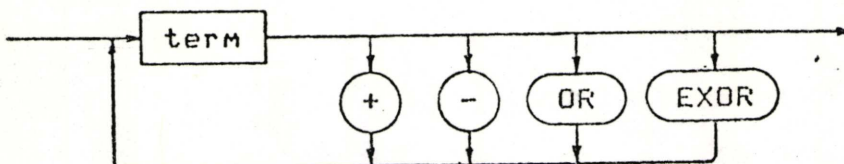
faktor



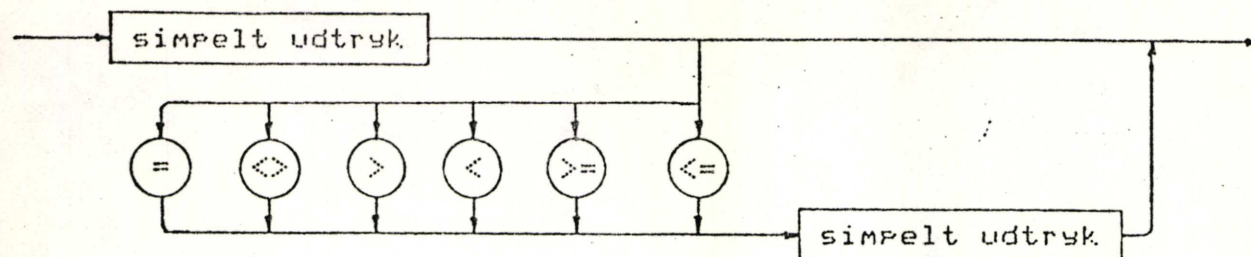
term



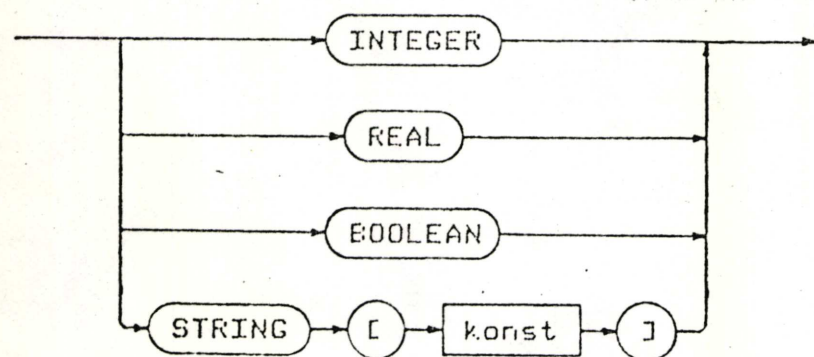
simpelt udtryk



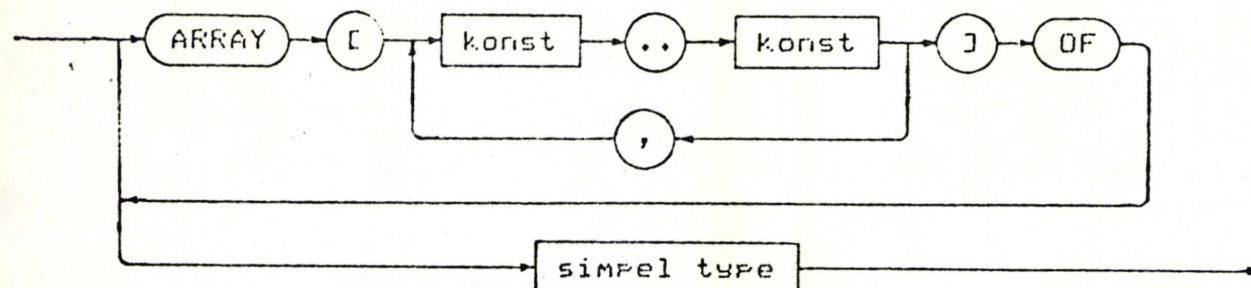
udtryk



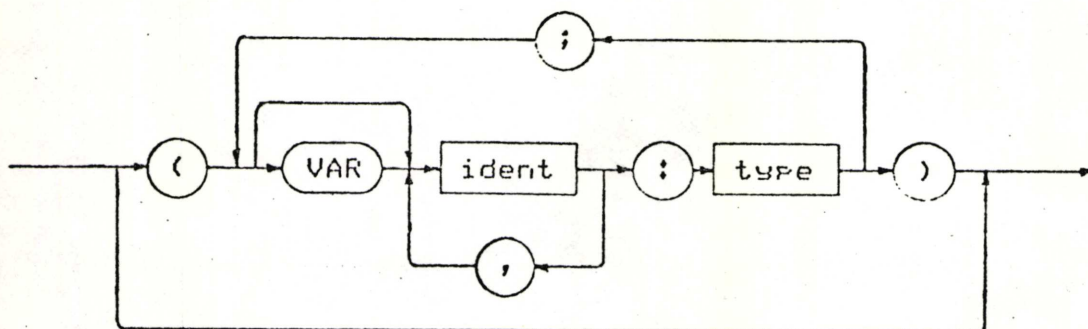
simpel type



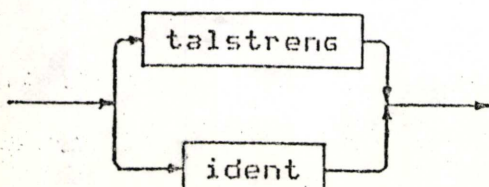
type

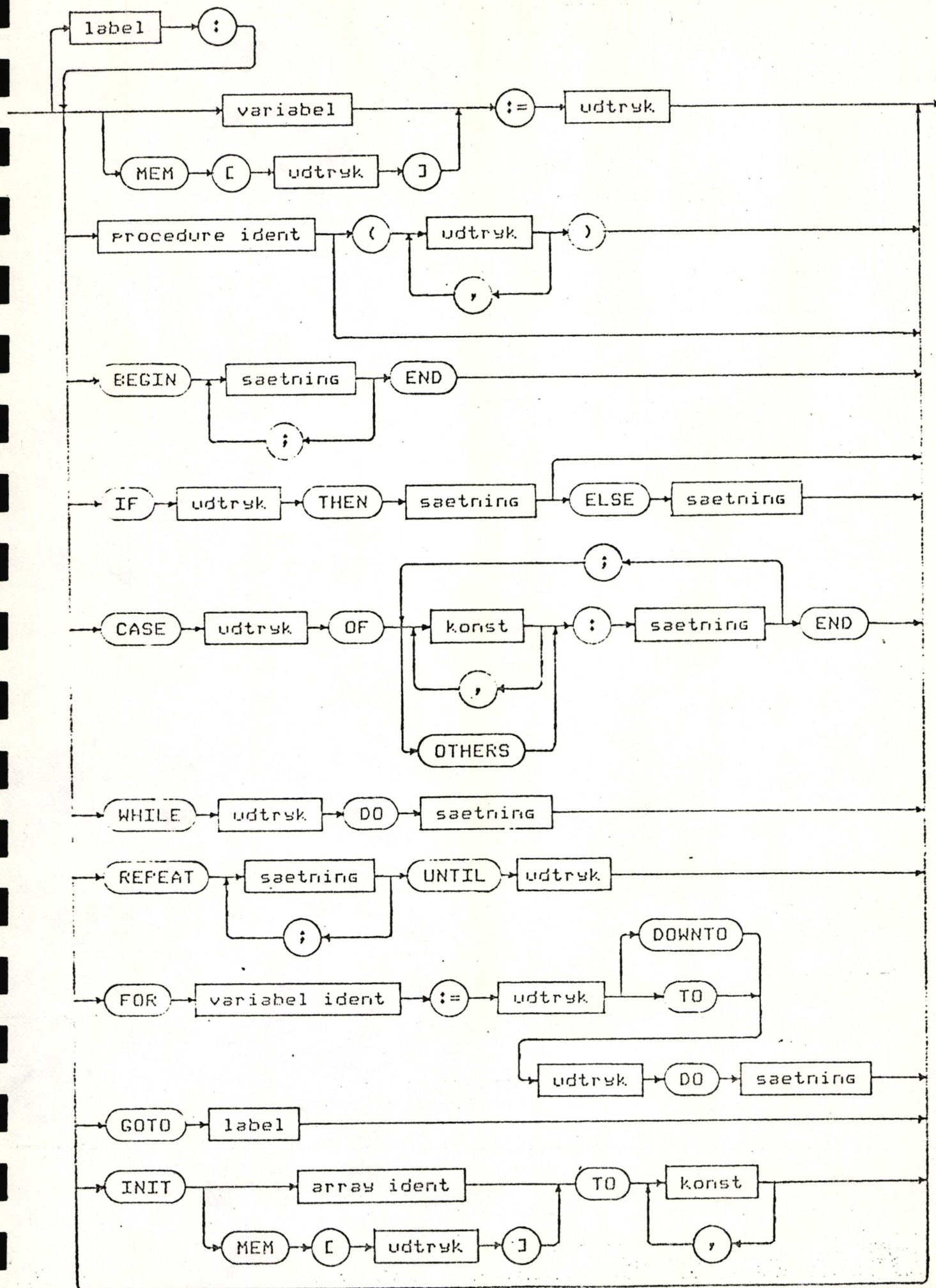


parameterliste

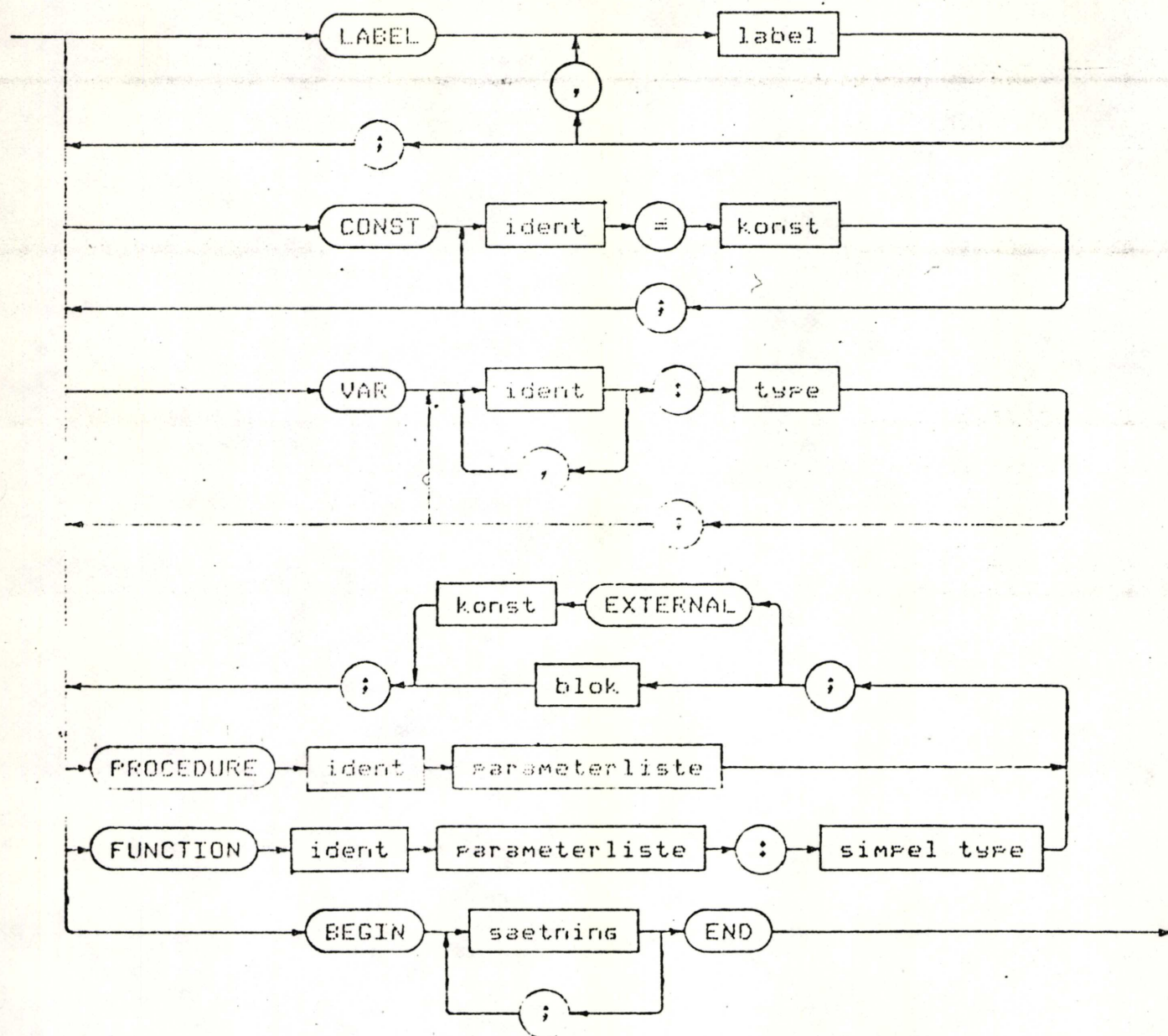


label

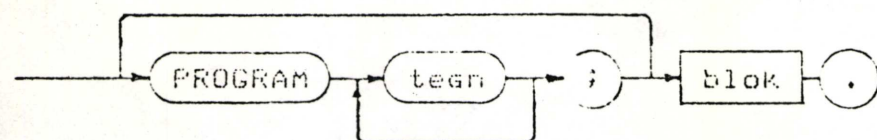




Block



PROGRAM



APPENDIX B -- EKSEMPLER PÅ UNDERPROGRAMMER

```

(* Funktionen value konverterer tallet i strenges s til et *)
(* reelt tal. Variablen p peger på det første tegn der skal *)
(* indgå i konverteringen. Efter kaldet peger p på det før- *)
(* ste tegn der ikke indgik i konverteringen *)

```

```

FUNCTION value(s: STRING(.48.)); VAR p: INTEGER): REAL;

```

```

CONST
  zero=48; (* ASCII zero *)

```

```

VAR
  r,f: REAL;
  ch: STRING(.1.);
  neg,decpoint: BOOLEAN;

```

```

PROCEDURE nextchar;
BEGIN
  ch:=mid(s,p,1); p:=succ(p)
END (* of nextchar *);

```

```

BEGIN
  f:=1; nextchar;
  IF ch='-' THEN
    BEGIN neg:=true; nextchar END;
    WHILE (ch>='0') AND (ch<='9') DO
      BEGIN
        r:=r*10+(ord(ch)-zero);
        IF decpoint THEN f:=f*10;
        nextchar;
        IF (ch='.') AND NOT decpoint THEN
          BEGIN decpoint:=true; nextchar END;
        END;
        p:=pred(p);
        IF neg THEN value:=-r/f ELSE value:=r/f
      END (* of value *);

```

```

(* Funktionen pos returnerer den første position i streng- *)
(* udtrykket s hvor strengudtrykket t forekommer. Hvis t *)
(* ikke findes i s returneres et 0. *)

```

```

FUNCTION pos(t,s: STRING(.48.)): INTEGER;
LABEL exit;

```

```

VAR
  ldif,lt,p: INTEGER;
BEGIN
  lt:=length(t); ldif:=length(s)-lt;
  WHILE p<=ldif DO
    BEGIN
      p:=succ(p);
      IF mid(s,p,lt)=t THEN
        BEGIN pos:=p; GOTO exit END
      END;
    exit:
  END (* of pos *);

```



```
(* Procedure topline placerer strengudtrykket s på skærmens *)  
(* øverste linie, og fjerner alle øvrige tegn. *)
```

```
PROCEDURE topline(s: STRING(.48.));  
CONST  
  toplineadr=$BC9; (* topliniens adresse -1 *)  
  space=32;        (* ASCII blanktegn *)  
VAR  
  p: INTEGER;  
BEGIN  
  FOR p:=1 TO length(s) DO  
    mem(.p+toplineadr.):=ord(mid(s,p,1));  
  FOR p:=p TO 48 DO  
    mem(.p+toplineadr.):=space  
END;
```


APPENDIX C -- SYSTEMARBEJDSLAGERET

Systemet bruger 128 bytes arbejdslager i området \$C80 til \$D00. I dette område kan følgende være af interesse for brugeren:

- C92-C93 WSP Programarbejdslagerets stackpointer. Når programmet startes, sættes WSP til programmets slutadresse. Hvergang en blok aktiveres (hovedprogrammet eller et underprogram), flyttes WSP længere op i lageret for at give plads til blokkens variable og parametre. Når blokken forlades, flyttes WSP tilbage til sin oprindelige værdi.
- C94-C95 PMTP Den øverste adresse i RAM-lageret, der må benyttes af programmet. Hvis WSP bliver større end eller lig med PMTP, stoppes programmet med runtime error 99.
- C98-C9B RNDN Randomgeneratorens sidst udregnede værdi. Brugeren kan starte en bestemt randomsekvens ved at lægge et fast tal ud i disse fire bytes.

Den første instruktionssekvens i et programs objektkode er et kald til systemets initialiseringsrutine, efterfulgt af 5 bytes parametre:

CD xx xx aa bb cc dd ee

Adressen bbaa er programmets slutadresse. WSP bliver sat lig denne værdi ved initieringen. Adressen ddcc angiver den øverste adresse i RAM-lageret, der må benyttes af programmet. PMTP bliver sat lig denne værdi ved initieringen. Byten ee afgør hvortil kontrollen skal overgives i tilfælde af en runtime fejl. Er ee nul, bliver kontrollen overgivet til sprogsystemet. Er ee forskellig fra nul, overgives kontrollen til NAS-SYS.

Området fra \$D00 til \$1000 bruges til systemstakken. Ved starten af et program bliver stackpointeren (SP) sat til \$1000. Om stakkens brug gælder der bl. a. følgende:

Et procedure- eller funktionskald optager 2 bytes.

En aktiv FOR-løkke optager 4 bytes.

Under evaluering af udtryk bruges stakken til at gemme mellemresultater. Ved sammenligning af strengudtryk kan der således optages indtil 512 bytes, hvis begge strenge er 255 tegn lange.

Under programkørsel bliver der IKKE checket på stackpointerens position. Brugeren bør derfor sikre sig, at rekursive underprogrammer ikke kalder sig selv uendeligt.

APPENDIX D -- INTERN DATAREPRESENTATION

Dette appendix er medtaget for at give brugeren en indsigt i, hvorledes BLS Pascal behandler de forskellige datatyper internt. I beskrivelsen skal symbolet 'adr' forstås som adressen på den første byte, en variabel af den beskrevne datatype optager i lageret. Det er denne værdi, 'standardfunktionen' addr returnerer.

INTEGER og BOOLEAN:

Internt behandles integers og booleans ens. En integer repræsenteres ved et 16 bit 2's komplement tal. I lageret gemmes en integer på følgende måde:

```
adr      Mindst betydende byte.
adr+1    Mest betydende byte.
```

Dette svarer til Z-80 processorens standard.

REAL:

En real repræsenteres ved en 40 bits mantisse og en 8 bits 2's exponent. I lageret gemmes en real på følgende måde:

```
adr      Mantissens mest betydende byte.
:
:
adr+4    Mantissens mindst betydende byte.
adr+5    2's exponent.
```

Exponenten er et binært tal med et offset på 80. Således svarer en exponent på 84 til, at mantissen skal ganges med $2^{(84-80)} = 2^4 = 16$. Hvis exponentens værdi er 80 opfattes hele tallet som værende 0. Værdien af mantissen kan opfattes som det fortegnsløse heltal, bestående af de 5 første bytes, divideret med 2^{40} . Mantissen er altid normaliseret, dvs. den mest betydende bit skal altid opfattes som værende sat. Fortegnet gemmes i den mest betydende bit: Er denne sat er tallet negativt, er den nul er tallet positivt.

STRING:

En string optager den maksimale længde plus 1 bytes i lageret. I den første byte står strengens nuværende længde. I den anden byte står strengens sidste tegn, i den tredje byte det næstsidste tegn, etc.:

```
adr      Længde (=n).
adr+1    Tegn nr. n.
adr+2    Tegn nr. n-1.
:
:
adr+n    Tegn nr. 1.
```

Hvis en streng ikke er fyldt helt ud, er indholdet af de resterende bytes ukendt.

ARRAYSTRUKTURER:

Et element i en arraystruktur har samme format som en enkeltvariabel af samme datatype. I lageret ligger elementet med det laveste index først (dvs. på den laveste adresse). Hvis arraystrukturen har flere dimensioner, tælles der op startende med den sidste dimension. En arraystruktur erklæret som:

```
a: ARRAY(.1..3,1..3.).
```

vil således blive gemt i den denne rækkefølge:

laveste adr.	a(.1,1.)
	a(.1,2.)
	a(.1,3.)
	a(.2,1.)
	a(.2,2.)
	:
	:
højeste adr.	a(.3,3.)

APPENDIX E -- EKSTERNE UNDERPROGRAMMER

Procedurer og funktioner erklæret med EXTERNAL specificationen giver brugeren mulighed for at kalde maskinkodesubrutiner. I pascalprogrammet behandles et EXTERNAL specificeret underprogram som et almindeligt underprogram.

Parameteroverførsel til et underprogram foregår via programarbejdslageret. Hvis underprogrammet er en funktion, afsættes der først plads til returværdien. Parametrene bliver derefter skubbet på arbejdsstakken en ad gangen. I en maskinkoderutine kan brugeren få tilgang til parametrene ved at indexere sig frem i forhold til programarbejdslagerets stackpointer WSP (se herom i APPENDIX C).

Value-parametre følger de regler, der er givet for de forskellige datatyper i APPENDIX D. Hvis en parameter er VAR specificeret, overføres adressen på den første byte i lageret, der optages af den aktuelle parameter. Hvis en var-parameter er en arraystruktur, overføres adressen på det første element.

Hvis en funktion fx. er erklæret som:

```
FUNCTION test(VAR i: INTEGER; r: REAL): STRING(.16.);
```

vil toppen af arbejdslageret se således ud ved kaldet:

laveste adr.	WSP-25	17 bytes til resultatet
	:	(nulstillede ved kaldet).
	:	
	WSP-9	
	WSP-8	Adressen på heltalsvariablen der
	WSP-7	udgør den aktuelle parameter.
	WSP-6	Værdi af typen real.
	:	
	:	
højeste adr.	WSP-1	

hvor WSP er den adresse der står i adresserne \$C92-\$C93 (se herom i APPENDIX C). Adressen på den første byte i resultatet kan fx. nås på følgende måde:

```
WSP: EQU 0C92H
      :
      :
      LD HL,(WSP)
      LD DE,-25
      ADD HL,DE
```

efter udførelsen af koden peger HL registret på den første byte af den lagerplads, der er sat af til strengresultatet. Adressen på heltalsvariablen kan fx. nås på følgende måde:

```
LD HL,(WSP)
LD DE,-8
ADD HL,DE
LD E,(HL)
INC HL
LD D,(HL)
EX DE,HL
```


Som et eksempel på brugen af EXTERNAL specificerede underprogrammer er vist en procedure til udlæsning på en dataport, og en funktion til indlæsning fra en dataport. I programteksten er følgende erklæringer nødvendige:

```
CONST
    outportadr=$D00;
    inportadr=$D0D;
:
:
PROCEDURE outport(port,data: INTEGER);
EXTERNAL outportadr;
:
FUNCTION inport(port: INTEGER): INTEGER;
EXTERNAL inportadr;
```

Maskinkoderutinerne kan fx. laves på følgende måde:

```
0001 0D00                ORG  0D00H
0002
0003 0C92                WSP:  EQU  0C92H
0004
0005 0D00 DD2A920C OUTP:  LD    IX,(WSP)
0006 0D04 DD7EFE          LD    A,(IX-2)
0007 0D07 DD4EFC          LD    C,(IX-4)
0008 0D0A ED79            OUT   (C),A
0009 0D0C C9              RET
0010
0011 0D0D DD2A920C INP:   LD    IX,(WSP)
0012 0D11 DD4EFE          LD    C,(IX-2)
0013 0D14 ED78            IN     A,(C)
0014 0D16 DD77FC          LD    (IX-4),A
0015 0D19 C9              RET
0016
0017                      END
```

I en maskinkoderutine kan brugeren frit benytte RAM-lageret over WSP som arbejdslager.

APPENDIX F -- BENCHMARKTESTS OG TIDER

På side F.2 er vist de 14 Pascal benchmarktests, som de er foreslået i Personal Computer World fra december 1980. Målingerne er foretaget på en NASCOM 2 (Z-80 microprocessor, 4 MHz med 1 waitstate). Til sammenligning med de opnåede tider er vist målinger foretaget på en Heathkit H-11A (LSI 11/2 16-bits processor) med UCSD Pascal. Alle tider er i sekunder:

TEST	BLS Pascal	H-11A
Magnifier	0.8	3.9
Forloop	8.6	42.8
Whileloop	23.0	40.1
Repeatloop	20.8	35.0
Litteralassign	11.7	50.0
Memoryaccess	15.1	52.0
Realarithmic	59.8	61.7
Realalgebra	52.5	40.6
Vector	62.2	102.9
Equalif	24.3	66.8
Unequalif	24.2	65.8
Parameters	6.6	26.4
Value	12.5	29.3
Reference	12.1	29.7

Det bør nævnes at UCSD Pascal regner reals med 6+ betydende cifre, mens BLS Pascal regner med 11+ betydende cifre.

APPENDIX G -- LITTERATURLISTE

Kathleen Jensen, Niklaus Wirth:
PASCAL user manual and report
Springer-Verlag

Anders Haraldsson:
Programmering i Pascal
Teknisk forlag A/S

I.R. Wilson, A.M. Addyman:
A Practical Introduction to Pascal
Springer-Verlag

Rodnay Zaks:
Introduction to PASCAL
Sybex

Niklaus Wirth:
Algorithms + Data Structures = Programs
Prentice-Hall

APPENDIX H -- COMPILER FEJLMEDDELELSER

- 00 Fejladresse fundet.
- 01 Syntaxfejl eller manglende ';'.
- 02 '=' forventet.
- 03 ':' forventet.
- 04 ' ' forventet.
- 05 ' ' forventet.
- 06 '(' forventet.
- 07 ')' forventet.
- 08 ',' forventet.
- 09 '.' forventet.
- 10 '..' forventet.
- 11 ':= ' forventet.
- 20 Nedre grænse større end øvre grænse i arraydeklaration.
- 21 Overflow i arraydeklaration.
- 22 'OF' mangler i arraydeklaration.
- 23 Ulovligt tegn i identifier.
- 24 Strenglængde må ikke være nul.
- 25 Ukendt datatype.
- 30 Konstant af typen INTEGER forventet.
- 31 Konstant af typen STRING forventet.
- 32 Konstant af typen REAL forventet.
- 33 Heltalskonstant ikke i området $0 \leq k \leq 255$.
- 40 'BEGIN' forventet.
- 41 'THEN' mangler i IF-sætning.
- 42 Valgudtryk skal være af typen INTEGER eller STRING.
- 43 'OF' mangler i CASE-sætning.
- 44 'END' mangler i CASE-sætning.
- 45 'DO' mangler i WHILE-sætning.
- 46 Variabel af typen INTEGER forventet.
- 47 'TO'/'DOWNT0' mangler i FOR-sætning.
- 48 'DO' mangler i FOR-sætning.
- 49 Labelidentificer er ikke erklæret.
- 50 'TO' mangler i INIT-sætning.
- 60 Typen STRING er ikke tilladt her.
- 61 Udtryk af typen INTEGER forventet.
- 62 Udtryk af typen STRING forventet.
- 63 Typeuoverensstemmelse i udtryk.
- 64 Ukendt identifier i udtryk.
- 65 Syntaxfejl eller overflow i talkonstant, eller strengkonstant indeholder et linieskift.
- 66 Strengkonstant for lang.
- 70 Typeuoverensstemmelse i tilskrivning eller parameterliste.
- 71 Ukendt variabelidentificer.
- 72 Ukendt arrayidentificer.
- 80 Label erklæret og refereret men ikke defineret.
- 99 Programmet afsluttet ulovligt.

APPENDIX I -- RUNTIME FEJLMEDDELELSER

- 01 Overflow på REAL regneoperation.
- 02 Division med 0 forsøgt.
- 03 Kvadratroden forsøgt udregnet med et negativt argument.
- 04 Den naturlige logaritme forsøgt udregnet med et argument der er mindre end eller lig med 0.
- 05 Reelt tal udenfor området $-32768 \leq R \leq 32767$ forsøgt konverteret til et heltal.

- 10 Den resulterende streng ved CONCAT har en længde der er større end 255, eller positionsnummeret ved MID er mindre end eller lig med 0.

- 20 Et af index ved en reference til et arrayelement er udenfor de tilladte grænser.

- 99 Workspace overflow. Alt tilgængeligt arbejdslager er opbrugt.