PICCOLINE

PolyPascal

Brugervejledning

PICCOLINE

PolyPascal
Brugervejledning

1. Pakke navn

SW1402 PolyPascal 3.11 release 1.2.

2. Pakke indhold

1 Diskette mærket SW1402
1 Ringbind indeholdende
- PolyPascal Brugermanual
- PolyPascal Programmeringsmanual
- PolyFile Referencemanual

3. Diskette indhold

Disketten indeholder følgende filer:

README.DOC	PPAS.CMD	PPAS.HLP	PPASRC.HLP
PPAS.ERM	PPBS.CMD	PP87.CMD	INSTALL.CMD
INSTALL.TRM	INSTALL.DAT	INSTALL.DOC	INSTALL.PAS
CALC.PAS	HEXDUMP.PAS	PRIMES.PAS	QSORT.PAS
LIST.PAS	CROSSREF.PAS	DATMAN.PAS	NPFKEY.PAS
ADDKEY.PAS	DELKEY.PAS	DATABASE.PAS	GSX.PAS
GSX.DOC	INSTJOB.SUB		

4. Ændringer i PolyPascal sammenlignet med COMPAS Pascal

PolyPascal hed tidligere COMPAS Pascal, og PolyPascal V3.11 er en direkte videreudvikling af COMPAS Pascal V3.07. PolyPascal er fuldt ud kildetekstkompatibel med COMPAS Pascal, bortset fra de følgende mindre forskelle (kapitelreferencerne henviser til PolyPascal Programmeringsmanualen):

- TEXT identifieren er et reserveret ord i PolyPascal.

- PolyPascal behandler logiske enheder (CON:, LST:, etc.) anderledes end COMPAS Pascal. Specielt gælder der, at "eof" og "eoln" opfører sig anderledes, når de anvendes på logiske enheder. I modsætning til COMPAS Pascal er der i PolyPascal ikke forskel på behandlingen af logiske enheder og diskfiler. Kapitel 13.3 indeholder yderligere information.

- Standardprocedurerne "blockread" og "blockwrite" kræver fire parametre i stedet for tre. Den fjerde parameter returnerer det antal records, der faktisk blev overført. Kapitel 13.4 indeholder yderligere information.

- PolyPascal åbner og lukker overlay-filer hver gang der læses fra dem. Som programmør behøver man derfor ikke længere bekymre sig om at lukke overlay-filer. Y compiler direktivet er afskaffet. I stedet kan standardproceduren "ovdrive" bruges til at angive hvilken disk, der indeholder overlay-filerne. Kapitel 15.9 indeholder yderligere information.

- Standardprocedurerne "chain" og "execute" sætter ikke længere et flag i adresse $80 i datasegmentet. Kapitel 20 indeholder yderligere informationer.

- Det interne format af filvariable (FIB'er) er ændret. Kapitel 23.1.5 indeholder yderligere informationer.

- To nye standardfunktioner er indført til at hente kommandolinieparametre: argcnt og argstr(n).

- Brugeren kan skrive sin egen fejlbehandlingsrutine, der bliver kaldt i tilfælde af en I/O eller kørselsfejl.

- En prædefineret variabel ved navn "cbreak" kan benyttes til at vælge, om PolyPascal skal checke for Ctrl-S og Ctrl-C eller ej.

- PolyPascal frigiver automatisk skriveren når et program stopper efter en RUN kommando. Ligeledes frigives skriveren automatisk når editorens Ctrl-KP kommando er færdig med at udskrive en blok.

- En standard heltalsvariabel ved navn "sysres" indeholder returkoden, der overføres i AX registeret ved fil-orienterede BDOS kald (så som open file, close file, read random, osv.).

4.1 Ændringer i forhold til PolyPascal 3.10, rel. 1.1

- Ved anvendelse ad "H. Partner/Piccoline-terminalen" i "Load a standard configuration" belastede PolyPascal maskinen, selv om programmet ventede på input.

- Fejl omkring filer/dynamiske variable rettet.

- Fejl omkring val rettet.

5. Installation

Fremgangsmåden ved installation afhænger af, om pakken skal installeres på en PICCOLINE, eller en Partner netvært, med 2 diskettestationer eller en Partner netvært med 1 diskettestation og 1 winchesterdisk.

2 diskette stationer:

- Start systemet som normalt (system diskette i station A). Hvis man havner i menu systemet, skal man trykke ESC for at komme til CCP/M-86 kommando niveauet (A> meldingen).

- Indsæt PolyPascal disketten i station B.

- Skift den aktuelle disk til station B med kommandoen:

  ```
  A>B:
  B>
  ```

- Overfør de nødvendige filer for at kunne udføre Poly-Pascal, fra distributionsdisken i station B til systemdisken i station A med kommandoen:

  ```
  B>SUBMIT INSTJOB A:
  ```

- Operationen er færdig når B> meldingen fremkommer på ny.

1 diskette station og 1 winchester disk

- Start systemet som normalt (system disk er winchester disk). Hvis man havner i menu systemet, skal man trykke ESC for at komme til CCP/M-86 kommando niveauet (B> meldingen).

- Indsæt PolyPascal disketten i station A.

- Skift den aktuelle disk til station A med kommandoen:

  ```
  B>A:
  A>
  ```

- Overfør de nødvendige filer for at kunne udføre Poly-Pascal, fra distributionsdisken i station A til systemdisken med kommandoen

  ```
  A>SUBMIT INSTJOB B:
  ```

- Operationen er færdig når A> meldingen fremkommer på ny.

Yderligere installation er ikke nødvendig.

5.1 Direkte skærm I/O

PolyPascal kan bringes til at skrive direkte i PICCOLINE's skærmbuffer, hvilket er betydelig hurtigere end almindelig udskrift. Hvis dette ønskes, skal man anvende følgende fremgangsmåde:

- Vælg "I. PICCOLINE High Speed"-terminalen i "Load a standard configuration"-menuen i INSTALL programmet.

- Bemærk, at funktionstasterne bliver programmeret på følgende måde:

F1 = Find (ctrl-QF)	F2 = Alter (ctrl-QA)
F3 = Mid screen (goto 1,12)	F4 = Start of text (ctrl-QR)
F5 = End of text (ctrl-QC)	F6 = Delete Line (ctrl-Y)
F7 = Delete word (ctrlT)	F8 = Adjust mode (ctrl-W)
F9 = Scroll down (ctrl-QXX)	F10= Scroll up (ctrl-QEE)
F11= Start of line (ctrl-QS)	F12= End of line (ctrl-Q)

Tast	Funktion
	Markør til venstre
	Markør til højre
	Markør en linie op
	Markør en linie ned
	Markør til toppen af skærmen
A1	Et ord til venstre
A2	Et ord til højre
A3	En side op
A4	En side ned
Tegn Ind	INSERT on/off
Slet tegn	Slet tegn til højre

PolyPascal
Brugervejledning

PolyPascal
Brugermanual

PolyPascal
Programmeringsmanual

PolyPascal
Reference Manual

PN: 99110090

Udgivet af A/S Regnecentralen af 1979, København

PolyPascal

Pascal Program Development System

Version 3.1

BRUGERMANUAL

PolyData MicroCenter A/S
Åboulevarden 13
DK–1960 København V

PN: 991 10 090

PolyData MicroCenter A/S
Åboulevarden 13
DK–1960 København V

Telefon: (01) 35 61 66
Telex: 27439 poly dk

Indholdsfortegnelse

0 Indledning 5

0.1 Systemkrav 5
0.2 Distributionsdisketten 6
0.3 Manualens notationer 7

1 Opstart og kommandolinier 9

1.1 Opstart af PolyPascal 9
1.2 Kommandolinier 10
1.3 HELP kommandoen 11

2 Læsning, skrivning og navngivning af filer 13

2.1 LOAD kommandoen 13
2.2 SAVE kommandoen 14
2.3 NAME kommandoen 14

3 Editoren 15

3.1 EDIT kommandoen 15
3.2 WHERE kommandoen 15
3.3 Editorkommandoer 16
3.3.1 Kommandoer til flytning af cursor 18
3.3.2 Kommandoer til valg af tilstand 19
3.3.3 Editeringskommandoer 19
3.3.4 Blokkommandoer 20
3.3.5 Søg/erstat kommandoer 21
3.3.6 ADJUST tilstanden 23
3.3.7 Andre editorkommandoer 23
3.4 Editorens fejlmeddelelser 24

4 Compileren 27

4.1 COMPILE kommandoen 27
4.2 RUN kommandoen 28
4.3 PROGRAM kommandoen 29
4.4 OBJECT kommandoen 31
4.5 FIND kommandoen 32
4.6 Fejlhåndtering 33

5 Andre kommandoer **35**

5.1 DIR kommandoen 35
5.2 USE kommandoen 35
5.3 MEMORY kommandoen 36
5.4 ZAP kommandoen 37
5.5 QUIT kommandoen 37

Index **39**

Indledning

PolyPascal er et dansk udviklet programudviklingssystem baseret på det blokstrukturerede programmeringssprog Pascal. PolyPascal findes i tre forskellige versioner: Til MS–DOS eller CP/M–86 baserede systemer med 8086 (eller 8088) processor og til CP/M–80 baserede systemer med Z–80 processor. 8086 versionen understøtter 8087 floating point co–processoren. I manualen kaldes 8086 versionen PolyPascal–86 og Z–80 versionen PolyPascal–80.

PolyPascal har alle de faciliteter, der behøves for editering, oversættelse, og kørsel af programmer skrevet i Pascal. Systemet består af en run–time programdel, en skærmorienteret editor og en Pascal compiler, og fylder ialt kun 34K bytes i 8086 versionen og 28K bytes i Z–80 versionen.

PolyPascal følger nøje definitionen af Standard Pascal, der gives af K. Jensen og N. Wirth i bogen "Pascal User Manual and Report". I PolyPascal findes desuden en del udvidelser, der yderligere forbedrer sproget.

Denne manual beskriver opstart og brug af PolyPascal systemet. I programmeringsspørgsmål henvises der til PolyPascal programmeringsmanualen.

PolyPascal systemet og den tilhørende dokumentation er forfattet af Anders Hejlsberg og Christen Fihl.

0.1 Systemkrav

PolyPascal–86 til MS–DOS og PC–DOS
Intel 8088, 8086, 80186 eller 80286 microprocessor. MS–DOS eller PC–DOS operativsystem, version 2.0 eller højere. Mindst 64K bytes RAM til rådighed når operativsystemet er indlæst. Mindst en diskettestation.

PolyPascal–86 til CP/M–86
Intel 8088, 8086, 80186 eller 80286 microprocessor. CP/M–86, MP/M–86 eller Concurrent CP/M–86 operativsystem. Mindst 64K bytes RAM til rådighed når operativsystemet er indlæst. Mindst en diskettestation.

PolyPascal–80 til CP/M–80
Zilog Z–80 microprocessor. CP/M operativsystem, version 2.2 eller højere. Mindst 48K bytes RAM til rådighed når operativsystemet er indlæst. Mindst en diskettestation.

0.2 Distributionsdisketten

De følgende filer findes på distributionsdisketten ('COM' typen anvendes af MS–DOS og CP/M–80. 'CMD' typen anvendes af CP/M–86):

README.DOC	En tekstfil der beskriver den nuværende version af PolyPascal systemet. Udskriv og læs denne fil, for eksempel ved hjælp af operativsystemets 'TYPE' kommando, før PolyPascal tages i brug.
PPAS.COM PPAS.CMD	Denne fil indeholder selve PolyPascal systemet, dvs. run–time biblioteket, den skærmorienterede editor, og Pascal compileren.
PPAS.HLP	Denne fil indeholder de hjælptekster, der vises af HELP kommandoen og ^J kommandoen i editoren.
PPAS.ERM	Denne fil indeholder de fejlmeddelelser, der bruges af compileren, når den rapporterer fejl under oversættelser.
PPBS.COM PPBS.CMD	Business versionen af PolyPascal (findes ikke i en 8–bits version og følger derfor ikke med PolyPascal–80). Business versionen anvender 10–byte reelle tal i BCD format, hvilket giver 18 betydende cifre. Envidere indeholder denne version er række avancerede formatteringsfunktioner til tal og strenge.

PP87.COM
PP87.CMD

8087 versionen af PolyPascal (findes ikke i en bits version og følger derfor ikke med PolyPascal- 80). Denne version anvender 8087 co-processoren til floating point kalkulationer, men er ellers fuld stændig magen til standardversionen.

INSTALL.COM
INSTALL.CMD

PolyPascal installationsprogram. Dette program bruges til at indlægge systemafhængige parametre i PolyPascal. Hvis PolyPascal er leveret som en uinstalleret version, skal dette program køres før systemet tages i brug. INSTALL kan desuden anvendes til at rette enkeltparametre i en præ-installeret version. INSTALL er selvforklarende. Bemærk, at Business versionen og 8087 versionen altid distribueres som uinstallerede versioner, og at de derfor må installeres førend de kan anvendes.

INSTALL.TRM
STALL.DAT

INSTALL programmets datafiler. Disse filer kan IN indeholde installationsdata for op til 40 forskellige computersystemer.

Desuden kan distributionsdisketten eventuelt indeholde en række demonstrationsprogrammer, der i givet fald vil være beskrevet i README.DOC filen.

0.3 Manualens notationer

I manualen skrives hextal (tal i 16-tals systemet) med et foranstillet '$' tegn, for eksempel $16E0.

Betegnelsen "arbejdsfil" anvendes om den tekst, der for nuværende findes i arbejdslageret. Arbejdsfilens navn anvendes automatisk i en lang række tilfælde når intet andet angives.

Betegnelsen "filnavn" angiver et MS-DOS eller CP/M disk fil navn. Formatet af et filnavn afhænger af, hvilken version af PolyPascal der anvendes.

MS–DOS versionen
I MS–DOS versionen er et filnavn faktisk en række af biblioteksnavne, der fører frem til det ønskede underbibliotek, efterfulgt af selve filnavnet:

<disk>:\<dirnavn>\...\<dirnavn>\<filnavn>

Hvis rækken starter med en baglæns skråstreg (eller på nogle maskiner et stort Ø), starter MS–DOS søgningen i diskens hovedbibliotek; i modsat fald starter søgningen i det nuværende underbibliotek.

<disk> er diskens identifier (A–O). Hvis <disk> og det efterfølgende kolon udelades, vælges den nuværende disk. <dirnavn> og <filnavn> er disk fil navne i formatet:

<navn>.<type>

hvor <navn> er op til otte bogstaver eller talcifre og <type> er op til tre bogstaver eller talcifre. Hvis punktummet og <type> feltet udelades, vælger systemet automatisk en filetype afhængig af den sammenhæng filnavnet anvendes i.

CP/M versionerne
Det generelle format af et CP/M filnavn er:

<disk>:<navn>.<type>

hvor <disk> er diskens identifier (A–P), <navn> er filnavnet (op til otte bogstaver eller talcifre), og <type> er filtypen (op til tre bogstaver eller talcifre). Hvis <disk> og det efterfølgende kolon udelades, vælges den nuværende disk. Hvis punktummet og <type> feltet udelades, vælger systemet automatisk en filtype afhængig af den sammenhæng filnavnet anvendes i.

1. Opstart og kommandolinier

1.1 Opstart af PolyPascal

PolyPascal opstartes fra operativsystemet med kommandolinien:

```
PPAS
```

Ved opstart udskriver systemet:

```
PolyPascal-XX V3.YZ (RRRRRR Standard Version)
Copyright (C) 1984 PolyData MicroCenter A/S
Include error messages (Y/N)?
```

hvor XX er 80 eller 86, YZ er revisionsnummeret, og RRRRRR er navnet på operativsystemet. Tast 'Y' for at indlæse filen med fejlmeddelelser (PPAS.ERM) eller ethvert andet tegn for at undgå indlæsningen. Hvis PPAS.ERM indlæses, vil compileren udskrive en fejlmeddelelse, når den finder en fejl under en oversættelse. I modsat fald udskrives kun fejlens nummer (fejlteksterne findes i appendix H i PolyPascal programmeringsmanualen).

Efter en opstart som beskrevet ovenfor, kan PolyPascal genstartes fra MS-DOS eller CP/M med kommandolinien:

```
PPAS *
```

Dette fordrer naturligvis, at der ikke i mellemtiden er kørt programmer, der har overskrevet dele af arbejdslageret.

1.2 Kommandolinier

Når PolyPascal er klar til at modtage og udføre en kommandolinie, udskrives en dobbelt vinkel ('>>'). Under indtastning af kommandolinier kan de følgende editeringsnøgler anvendes:

BS	Sletter det sidst indtastede tegn. Denne funktion findes normalt på tastaturet som BS, BACK, BACKSPACE eller en venstrepil, men den kan altid frembringes ved at taste Ctrl–H.
ESC	Sletter hele linien.
Ctrl–X	Udfører samme funktion som ESC.
Ctrl–D	Genkalder et tegn fra den sidst indtastede linie.
Ctrl–R	Genkalder hele den sidst indtastede linie.
RETURN	Afslutter indtastningen. Denne funktion findes normalt på tastaturet som RETURN eller ENTER.

Nedenfor vises de kommandoer, der findes i PolyPascal (hver enkelt kommando beskrives fuldt ud i de følgende kapitler):

HELP	Udskriv hjælpetekster.
LOAD	Indlæs en ny fil.
SAVE	Gem arbejdsfilen.
NAME	Vis/ret arbejdsfilens navn.
EDIT	Start editoren.
COMPILE	Oversæt arbejdsfilen og gem koden i lageret.
RUN	Start kørsel af program.
PROGRAM	Oversæt kildetekst og gem koden i en programfil.
OBJECT	Oversæt kildetekst og gem koden i objektfil.
FIND	Find kørselsfejl i kildetekst.
WHERE	Find compilerfejl i include–fil.
DIR	Udskriv en disks bibliotek.
USE	Vis/ret nuværende disk/underbibliotek.
MEMORY	Udskriv størrelse af kildetekst og arbejdslager.
ZAP	Slet kildetekst eller disk fil.
QUIT	Returner til operativsystemet.

Bemærk, at alle kommandoer må forkortes til deres første bogstav.

1.3 HELP kommandoen

HELP kommandoen bruges til at vise hjælpetekster. Kommandolinies format er:

HELP <kommando>

hvor <kommando> er en af de ovenstående kommandoer (eller blot det første bogstav). Hvis <kommando> ikke angives, udskriver HELP en generel kommandooversigt. Ellers udskrives en komplet beskrivelse af den pågældende kommando.

HELP kommandoen virker kun hvis PPAS.HLP filen findes på disken. Hvis PPAS.HLP mangler udskriver HELP kommandoen:

Help file not found

Hvis den kommando der angives ikke findes, udskriver HELP:

No such help text

2. Læsning, skrivning og navngivning af filer

2.1 LOAD kommandoen

LOAD kommandoen bruges til at indlæse en ny kildetekstfil i arbejdslageret. Kommandoliniens format er:

LOAD <filnavn>

Systemet vælger automatisk '.PAS' hvis filtypen udelades. Hvis <filnavn> udelades, anvendes arbejdsfilens navn. Hvis der er blevet rettet i arbejdsfilen siden den blev indlæst eller sidst gemt, spørger PolyPascal:

<arbejdsfil> not saved. Save (Y/N)?

Hvis der svares 'Y' bliver arbejdsfilen gemt førend den nye fil indlæses. Hvis der svares med 'N' bliver arbejdsfilen ikke gemt, og de ændringer, der eventuelt måtte være foretaget i den, går tabt. Hvis der svares med andet, afbrydes LOAD kommandoen.

Når en fil indlæses, bliver den udnævnt til arbejdsfil, og dens navn anvendes automatisk af LOAD, SAVE, PROGRAM og OBJECT kommandoerne når intet andet angives.

Hvis den fil, der angives, ikke findes udskriver systemet:

No such file

Hvis der ikke er plads i arbejdslageret til at indlæse filen, udskriver systemet:

File too big

I begge disse tilfælde nulstilles arbejdslageret, og arbejdsfilens navn bliver 'WORK.PAS'.

2.2 SAVE kommandoen

SAVE kommandoen bruges til at lagre arbejdsfilen på disk. Kommandoliniens format er:

SAVE <filnavn>

Systemet vælger automatisk '.PAS' hvis filtypen udelades. Hvis <filnavn> udelades, anvendes arbejdsfilens navn. Hvis der allerede findes en fil af samme navn og type på disken, bliver denne fils type rettet til '.BAK' førend den nye fil oprettes (INSTALL programmet kan eventuelt anvendes til at deaktivere denne facilitet). Hvis filnavnet er ulovligt, udskriver PolyPascal:

Invalid filename

Hvis diskens bibliotek er fuldt, udskriver PolyPascal:

Directory is full

Hvis disken er fuld, udskriver PolyPascal:

Disk is full

Hvis en af de ovenstående fejl rapporteres, så indsæt en anden disk, udfør en USE kommando (kun nødvendigt i CP/M versionerne) og prøv derefter SAVE igen.

2.3 NAME kommandoen

NAME kommandoen anvendes til at vise og eventuelt ændre arbejdsfilens navn. Arbejdsfilens navn bruges af LOAD, SAVE, PROGRAM og OBJECT kommandoerne hvis intet andet angives. Kommandoliniens format er:

NAME <filnavn>

Systemet vælger automatisk '.PAS' hvis filtypen udelades. Hvis <filnavn> udelades, bliver arbejdsfilens navn blot udskrevet; i modsat fald ændres arbejdsfilens navn til <filnavn>.

Arbejdsfilens navn sættes til 'WORK.PAS' ved opstart af PolyPascal, og når en ZAP kommando udføres. INSTALL programmet kan eventuelt anvendes til at ændre dette filnavn.

3. Editoren

PolyPascal editoren anvendes til indtastning og editering af programtekster. Den maksimale størrelse af en kildetekst i PolyPascal–80 er kun begrænset af det til rådighed værende RAM lager (op til 35K bytes afhængig af maskinen). Den maksimale størrelse af en kildetekst i PolyPascal–86 er ca. 60K bytes, men kan dog være mindre på maskiner med meget lidt RAM lager. Hvis et program bliver for stort til at være i editorens buffer på en gang, må det deles ned i mindre moduler, der sammenkædes under oversættelsen ved hjælp af include–filer.

3.1 EDIT kommandoen

EDIT kommandoen bruges til at starte editoren. Kommandoliniens format er:

EDIT <filnavn>

Systemet vælger automatisk '.PAS' hvis filtypen udelades. Hvis <filnavn> udelades (hvilket det normalt gør), bliver editoren startet umiddelbart og man kan da editere arbejdsfilen. Hvis <filnavn> angives, bliver arbejdsfilen gemt (hvis der er rettet i den) og den nye fil indlæst før editoren startes. Hvis man i dette tilfælde forlader editoren med en ^K^D kommando, udfører PolyPascal den modsatte proces, dvs. gemmer den nye fil og genindlæser den oprindelige arbejdsfil. Hvis man derimod forlader editoren med en ^K^X kommando, bliver den nye fil udnævnt til arbejdsfilen.

3.2 WHERE kommandoen

WHERE kommandoen bruges til at genstarte editoren. Når editoren forlades (med en ^K^D eller ^K^X kommando), husker systemet på cursorens position i teksten. Hvis man senere udfører en WHERE kommando, genstartes editoren, og cursoren flyttes til det sted, man tidligere forlod.

Når compileren rapporterer en fejl, sørger den samtidig for, at en WHERE kommando vil flytte cursoren til det fejlramte sted. Efter en compilering med fejl, vil en WHERE kommando altså starte editoren og stille cursoren ved fejlen.

EDIT kommandoen vil, i modsætning til WHERE kommandoen, altid placere cursoren i begyndelsen af teksten. WHERE kommandoen tillader ikke, som i EDIT kommandoen, at man angiver et filnavn.

3.3 Editorkommandoer

Editoren er skærm–orienteret, dvs. den er specielt designet til brug på en skærm–terminal i modsætning til mange linie–orienterede editorer. Når editoren startes, udskrives den tekst, der ligger i lageret, på skærmen. Hvis teksten er for lang til at blive vist på skærmen i sin fulde udstrækning på en gang (hvilket den normalt er), vises kun den første del. Skærmen fungerer således som et "vindue" til teksten – hele teksten ligger i lageret og kan umiddelbart behandles, men skærmen kan kun vise en del af den ad gangen. Når man, ved hjælp af en eller flere kommandoer, flytter til et sted i teksten, der ikke vises på skærmen, flytter skærmen med, og viser i stedet et andet udsnit.

Linierne i en tekst kan være lige så lange man ønsker det. Hvis en linie er længere end skærmens bredde, bliver der vist et '+' tegn i den yderste position.

Cursoren markerer en position i teksten og kan flyttes til ethvert sted på skærmen, der optages af teksten. Cursoren kan ikke flyttes "udenfor" skærmen. Prøver man det, flytter skærmen i stedet med, og viser et nyt udsnit af teksten. Skærmen kan ikke alene rulle op og ned; den kan også rulle sidelæns, således at man får mulighed for at rette meget lange linier. INSTALL programmet kan anvendes til at justere den skridtstørrelse der anvendes, når skærmen ruller sidelæns.

Når man bruger editoren opdager man hurtigt, at den er ganske intelligent. For eksempel opdaterer den kun skærmen når den har tid, dvs. når der ikke indtastes tegn eller kommandoer.

Tegn indsættes i teksten blot ved at taste dem. De nye tegn bliver enten indføjet (hvorved resten af linien flyttes en position mod højre) eller også erstatter de de tidligere tegn, afhængig af hvilken tilstand editoren befinder sig i.

Editoren styres ved hjælp af kontroltegn. Et kontroltegn frembringes på tastaturet ved at trykke CONTROL (eller KODE eller CTRL) tasten og et bogstav samtidig. I denne manual anvendes en op–pil umiddelbart foran et tegn til at indikere et kontroltegn. For eksempel anvendes ^A for Ctrl–A.

Da editoren har flere kommandoer end det antal kontroltegn, der kan frembringes på et tastatur, består nogle editorkommandoer af to tegn, for eksempel ^K^D, hvilket betyder Ctrl–K efterfulgt af Ctrl–D. Bemærk, at det andet tegn i en sådan sekvens ikke behøver at indtastes som et kontroltegn. Det Ctrl–D tegn der indgår i ^K^D kommandoen kan både indtastes som Ctrl–D, 'D' eller 'd'.

PolyPascal systemets INSTALL program giver mulighed for at definere alternative sekvenser for samtlige editorkommandoer, således at man kan udnytte ethvert tastaturs specielle nøgler. Et typisk eksempel er at definere tastaturets cursorpile til at udføre samme funktion som standard cursorkontrollerne (^S, ^D, ^E og ^X). Når manualen refererer til en af standardtasterne, kan man i stedet anvende den alternative tast. INSTALL programmet kan også bruges til at vise en liste over de alternative taster, der for øjeblikket er defineret.

Den øverste linie på skærmen er statuslinien, der til enhver tid viser editorens tilstand:

```
Line n Col n   Used n Free n   INS AUT TAB ADJ
```

Statusliniens felter er forklaret nedenfor.

Line n — Nummeret på den linie cursoren står i, talt fra starten af teksten.

Col n — Nummeret på den position cursoren står i, talt fra starten af linien.

Used n — Størrelsen af teksten i bytes.

Free n — Antallet af frie bytes.

INS — Vises når editoren er i INSERT mode. Når INSERT er til bliver indtastede tegn indføjet i teksten, hvorved resten af den nuværende linie skubbes mod højre. Når INSERT er fra, vil nye tegn i stedet erstatte de gamle.

AUT — Vises når editoren er i AUTO mode. Når AUTO er til, laver editoren automatisk en indrykning hvergang en ny linie startes. Indrykningen svarer altid til indrykningen på linien ovenover, men den kan let justeres ved hjælp af SPACE og BACKSPACE tasterne.

TAB Vises når editoren er i TAB mode. Når TAB er til, indsætter ^I kommandoen (TAB tasten) ASCII TAB tegn i teksten. I modsat fald bliver tabulatortegn omdannet til blanktegn. Tabulatoren stopper altid ved hver ottende position.

ADJ Vises når editoren er i ADJUST mode. ADJUST tilstanden bruges til hurtigt at justere indrykningen af en eller flere linier. Se herom i afsnit 3.3.6.

3.3.1 Kommandoer til flytning af cursor

^S Flytter cursoren en position mod venstre.

^D Flytter cursoren en position mod højre.

^A Flytter cursoren et ord mod venstre.

^F Flytter cursoren et ord mod højre.

^Q^S Flytter cursoren til begyndelsen af linien.

^Q^D Flytter cursoren til slutningen af linien.

^E Flytter cursoren en linie op.

^X Flytter cursoren en linie ned.

^Q^E Flytter cursoren til toppen af skærmen.

^Q^X Flytter cursoren til bunden af skærmen.

^R Flytter cursoren en side op.

^C Flytter cursoren en side ned.

^Q^R Flytter cursoren til begyndelsen af teksten.

^Q^C Flytter cursoren til slutningen af teksten.

3.3.2 Kommandoer til valg af tilstand

^V INSERT til/fra. Når INSERT er til, bliver indtastede tegn indføjet i teksten, hvorved resten af den nuværende linie skubbes mod højre. Når INSERT flaget er fra, vil nye tegn i stedet erstatte de gamle.

^Z AUTO til/fra. Når den automatiske tabulator er aktiv vil hver ny linie automatisk starte med den samme indrykning som linien ovenover.

^B TAB til/fra. Når TAB er til, indsætter ^I kommandoen (TAB tasten) ASCII TAB tegn i teksten. I modsat fald bliver tabulatortegn omdannet til blanktegn. Tabulatoren stopper altid ved hver ottende position. Ved brug af TAB tegn kan man spare lager, men ADJUST faciliteten kan ikke justere indrykningen af linier med ASCII TAB tegn.

^W ADJUST til/fra. ADJUST tilstanden bruges til hurtigt at justere indrykningen af en eller flere linier. Se herom i afsnit 3.3.6.

3.3.3 Editeringskommandoer

^M Ny linie (RETURN eller ENTER tasten). Denne kommando er afhængig af om INSERT er til eller fra. Når INSERT er til bliver der indsat et linieskift i teksten, hvilket giver en blank linie eller deler den nuværende linie i to hvis cursoren ikke er ved slutningen af linien. Cursoren stilles i begyndelsen af den nye linie. Hvis AUTO også er til, vil den nye linie starte med en indrykning der svarer til linien ovenover. Hvis INSERT er fra flyttes cursoren blot til begyndelsen af den næste linie.

^N Indsæt linieskift. Indsætter et linieskift ved cursoren, hvorved en blank linie fremkommer eller hvorved den nuværende linie bliver delt i to, hvis cursoren ikke er ved slutningen af linien. Cursoren bliver stående.

^I Tabuler (TAB tasten). Hvis TAB er til bliver et ASCII TAB tegn indsat i teksten. Hvis TAB er fra, bliver der indsat blanktegn, indtil cursoren er i en position, der er et multiplum af 8. Bemærk at TAB bør være fra hvis linierne i teksten senere skal kunne justeres i ADJUST mode.

^H Slet til venstre. Sletter tegnet til venstre for cursoren og flytter cursoren et tegn mod venstre.

^G Slet til højre. Sletter tegnet under cursoren. Cursoren bliver stående.

^T Slet et ord. Sletter det ord der står under og til højre for cursoren. Efterfølgende blanktegn bliver slettet. Et ord er en samling af tegn. Cursoren bliver stående.

^Y Slet linie. Sletter hele den linie cursoren befinder sig i, og flytter resten af skærmen en linie op. Cursoren flytter til begyndelsen af den næste linie.

^Q^H Slet til begyndelsen af linien. Sletter alle tegn til venstre for cursoren på den nuværende linie.

^Q^Y Slet til slutningen af linien. Sletter tegnet under cursoren og alle tegn til højre for den på den nuværende linie.

3.3.4 Blokkommandoer

Blokkommandoerne arbejder på blokke af tekst. En blok afgrænses af et start blok mærke og et slut blok mærke. Mærkerne kan ikke ses på skærmen – i stedet bliver alle tegn mellem mærkerne vist i forøget intensitet (INSTALL programmet kan anvendes til af definere de attributer, der anvendes til at vise tegn i blokke).

^K^B Sæt start blok mærke. Denne kommando sætter start blok mærket ved cursorens position, og fjerner ved samme lejlighed de tidligere mærker. Indtil slut blok mærket sættes, fungerer cursoren som den anden afgrænsning, så når cursoren flyttes, bliver alle tegn mellem denne og start blok mærket vist i forøget intensitet. Start blok mærket behøver ikke stå før slut blok mærket. Editoren finder selv ud af hvilket mærke, der kommer først.

^K^K Sæt slut blok mærke. Denne kommando sætter slut blok mærket ved cursorens position. Kommandoen bliver ignoreret hvis start blok mærket ikke er sat. Husk at cursoren automatisk fungerer som slut blok mærke når først start blok mærket er sat. I mange tilfælde er det derfor ikke nødvendigt at anvende ^K^K kommandoen.

^K^V Flyt blok. Flytter den afmærkede blok til cursorens nuværende position, dvs. kopierer blokken til den nuværende position og fjerner den fra den tidligere. Blokmærkerne fjernes.

^K^C Kopier blok. Kopierer den afmærkede blok til cursorens nuværende position. Blokmærkerne fjernes ikke.

^K^Y Slet blok. Sletter den afmærkede blok samt blokmærkerne.

^K^P Udskriv blok. Udskriver den afmærkede blok på printeren.

^K^R Læs blok fra fil. Denne kommando bruges til at indføje indholdet af en fil i teksten. Filen indføjes ved cursorens nuværende position. Kommandoen spørger efter navnet på den fil, der skal indlæses. Systemet vælger automatisk '.PAS' hvis filtypen udelades.

^K^W Skriv blok til fil. Denne kommando bruges til at lagre den afmærkede blok i en fil. Kommandoen spørger efter navnet på den fil, blokken skal lagres i. Systemet vælger automatisk '.PAS' hvis filtypen udelades. Filer af samme navn som det, der angives, får deres type ændret til '.BAK', førend den nye fil oprettes.

^K^H Fjern blokmærker. Bemærk, at blokmærkerne automatisk slettes, når editoren forlades.

3.3.5 Søg/erstat kommandoer

Søg/erstat kommandoerne bruges til hurtigt at finde og eventuelt erstatte forekomster af en given streng i teksten. Bemærk, at søgestrengen og erstatningsstrengen ikke kan indeholde linieskift (CR eller LF tegn), og at den maksimale længde for disse strenge er 32 tegn.

^Q^F Søg streng. Ved indtastning af denne kommando, spørger editoren om den streng, der skal findes ('Find?'). Indtast da søgestrengen og afslut med et tryk på RETURN. Eventuelle fejlindtastninger kan rettes med ^H kommandoen (BACKSP), og kontroltegn kan indføjes ved hjælp af ^P præfix kommandoen. Når søgestrengen er indlæst, spørger systemet om eventuelle tilvalg ('Options (G,N,U,W)?'). Mulige tilvalg er:

U = Ignorer store/små bogstaver. Dette tilvalg gør, at der ikke skelnes mellem store og små bogstaver i søgningen.
W = Find kun hele ord. Dette tilvalg gør, at søgestrengen ikke må være en del af et andet ord.

Eventuelle tilvalg skal indtastes uden blanktegn eller kommaer imellem, og linien med tilvalg skal afsluttes med RETURN. Søgningen starter efter tilvalgene er indlæst. Hvis søgestrengen findes, flyttes cursoren til den første position umiddelbart efter strengen. I modsat fald bliver cursoren stående. Søgningen omfatter kun teksten efter cursoren. Hvis hele teksten skal gennemsøges må man for eksempel udføre ^Q^R før ^Q^F.

^Q^A Søg og erstat streng. Denne kommando er en udvidet version af ^Q^F, idet den ydermere giver mulighed for at erstatte de fundne strenge med en anden streng. Kommandoen spørger efter en søgestreng ('Find?'), en erstatningsstreng ('Replace with?') og eventuelle tilvalg ('Options (G,N,U,W)?'). Mulige tilvalg er:

G = Global søgning. Fortsætter indtil hele teksten (efter cursorens startposition) er blevet gennemsøgt.
N = Erstat uden at spørge. 'Replace (Y/N)?' spørgsmålet udelades, og udskiftning foretages med det samme.
U = Ignorer store/små bogstaver. Dette tilvalg gør, at der ikke skelnes mellem store og små bogstaver i søgningen.
W = Find kun hele ord. Dette tilvalg gør, at søgestrengen ikke må være en del af et andet ord.

Hvis 'N' tilvalget ikke specificeres, vil kommandoen spørge 'Replace (Y/N)?' hvergang den finder en streng, og cursoren til skiftevis bevæge sig mellem teksten og spørgsmålet. Hvis der tastes 'Y', bliver strengen erstattet, hvorimod den forbliver uændret, hvis der tastes 'N'. Hvis 'G' tilvalget blev specificeret, stopper søgningen ikke efter den første forekomst af søgestrengen. I stedet fortsættes til hele teksten (efter cursorens startposition) er blevet gennemsøgt. Man kan dog afbryde søgningen ved at taste RETURN ved 'Replace (Y/N)?'.

^L Fortsæt søgning. Gentager den sidste ^Q^F eller ^Q^A kommando med de samme parametre.

3.3.6 ADJUST tilstanden

ADJUST tilstanden gør det nemt at justere indrykningen på en eller flere linier. ADJUST tilstanden startes ved at trykke ^W.

Når editoren definder sig i ADJUST tilstanden, bruges ^D og ^S til at flytte hele den nuværende linie mod højre eller venstre, og hvis cursoren flyttes op eller ned, ved hjælp af ^E eller ^X, gentages justeringen på den nye linie. Bemærk, at når man først har valgt en retning (enten op eller ned), kan man ikke gå tilbage i den modsatte retning.

Når linien (eller gruppen af linier) er justeret til den ønskede indrykning, kan ADJUST tilstanden forlades ved igen at trykke ^W.

Bemærk, at ADJUST tilstanden kan ikke justere linier, der indeholder tabulatortegn (ASCII TAB tegn). Hvis man skal kunne justere linierne i en tekst med ADJUST, bør teksten derfor indtastes med TABS fra (^B kommandoen).

3.3.7 Andre editorkommandoer

^J Hjælp. Viser en oversigt over alle editorens kommandoer på skærmen. Kommandoen virker kun, hvis hjælpetekstfilen findes på den disk, hvorfra PolyPascal blev startet. PolyPascal kan dog installeres til at søge efter hjælpetekstfilen på en anden disk.

^P Præfix til kontroltegn. Ved at taste ^P foran et kontroltegn, er det muligt at indføje dette kontroltegn i teksten. Hvis man for eksempel ønsker at indføje et ^L tegn (ASCII form–feed) i teksten, taster man ^P efterfulgt af ^L. Normalt vises kontroltegn som store bogstaver i inverteret skrift. Det er dog muligt at ændre dette ved hjælp af INSTALL programmet.

^K^D Afslut editering. Editoren afsluttes og der returneres til kommandoniveauet ('>>'). Hvis der blev angivet et filnavn ved start af editoren, eller hvis editoren blev startet af compileren på grund af en fejl i en include–fil, bliver filen gemt og den oprindelige fil indlæst igen. Læs mere herom i afsnit 4.6.

^K^X Forlad editoren. Normalt virker denne kommando nøjagtig som ^K^D, men hvis der blev angivet et filnavn ved start af editoren, eller hvis editoren blev startet af compileren på grund af en fejl i en include–fil, genindlæser denne kommando ikke den oprindelige fil.

3.4 Editorens fejlmeddelelser

Editorens fejlmeddelelser vises altid på skærmens øverste linie (der hvor statuslinien normalt står). Et eksempel:

ERROR: No room to insert. Press <RETURN>

Fejlmeddelelsen fjernes igen ved et tryk på RETURN tasten. Mulige fejlmeddelelser er:

No room to insert

Denne fejlmeddelelse vises hvis man prøver at udvide teksten når der ikke er mere plads i arbejdslageret.

No block is marked

Denne fejlmeddelelse vises, hvis man prøver at starte en blokkommando, når der ikke er afmærket en blok.

Help file not found

Denne fejltekst vises hvis ^J (hjælp) kommandoen ikke er i stand til at finde filen med hjælpetekster.

No such help text

Denne fejltekst vises hvis hjælpetekstfilen ikke indeholder en oversigt over editorens kommandoer. Normalt fremkommer denne tekst aldrig.

No such file

Denne fejltekst vises hvis man i ^K^R kommandoen angiver en fil der ikke findes.

File too big

Denne fejltekst vises hvis den fil, man prøver at indlæse med ^K^R kommandoen, er for stor.

Invalid filename

Denne fejltekst vises hvis man angiver et ulovligt filnavn.

Directory is full

Denne fejltekst vises hvis der ikke er plads til at oprette nye filer i diskens bibliotek.

Disk is full

Denne fejltekst vises hvis der ikke er plads til at gemme en fil på disken.

4. Compileren

Compileren er hjertet i PolyPascal systemet. Ved hjælp af denne oversættes PolyPascal programmer til maskinkode.

Når compileren startes fra en COMPILE eller en RUN kommando, bliver objektkoden gemt direkte ud i lageret. På denne måde arbejder compileren uhyre hurtigt (op til 5000 linier pr. minut), og efter endt compilering kan programmet startes umiddelbart. PolyPascal–80 brugere bør bemærke, at store programmer højst sansynligt ikke kan oversættes på denne måde, da både kildetekst og objektkode skal være til stede i lageret på en gang. Dette gælder også PolyPascal–86 brugere med begrænset RAM lager.

Når compileren startes fra en PROGRAM eller en OBJECT kommando, bliver objektkoden gemt ud i en diskfil. Denne metode er naturligvis noget langsommere end den ovenstående, men den kræver til gengæld ikke så meget arbejdslager, og muliggør desuden generering af programfiler ('.COM' eller '.CMD' filer), der kan startes fra operativsystemet blot ved at taste deres navn.

Ved hjælp af FIND kommandoen kan compileren bruges til at finde det sted i kildeteksten, der var skyld i en kørselsfejl i objektkoden. Denne facilitet er en uvurderlig hjælp under fejlsøgning af programmer.

4.1 COMPILE kommandoen

Når compileren startes fra en COMPILE kommando, bliver objektkoden gemt direkte ud i arbejdslageret, umiddelbart efter kildeteksten. Bemærk, at koden, der genereres af COMPILE kommandoen, slettes, når editoren startes. COMPILE kommandoen afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
Når compileren starter, udskriver den:

```
Compiling
```

Efter endt oversættelse udskriver systemet størrelsen af kode- og datasegmentet, samt antallet af ledige bytes (staksegmentet):

```
Code: rrrrrr bytes(aaaa paragraphs)
Data: ssssss bytes(bbbb paragraphs)
Free: tttttt bytes(cccc paragraphs)
```

De i parantes viste tal er størrelsen af de pågældende segmenter i "paragraffer". Tallene er i hexnotation, og en paragraf svarer til 16 bytes. Størrelsen af kodesegmentet omfatter både run-time biblioteket og selve programkoden.

PolyPascal-80

Når compileren starter, udskriver den:

```
Compiling
```

Efter endt oversættelse udskriver systemet størrelsen af objektkoden og datalageret, samt antallet af ledige bytes:

```
Code: rrrrr bytes(aaaa-bbbb)
Free: sssss bytes(cccc-dddd)
Data: ttttt bytes(eeee-ffff)
```

De i parentes viste tal er start- og slutadresserne på det givne område (i hexnotation). Størrelsen af objektkoden omfatter ikke run-time biblioteket.

4.2 RUN kommandoen

RUN kommandoen bruges til at starte kørsel af programmet. Hvis der ikke findes en objektkode, når kommandoen gives, bliver compileren først startet for at oversætte kildeteksten. Forudsat at der ikke bliver fundet en fejl under oversættelsen, eller hvis objektkoden allerede er tilstede, udskriver systemet:

```
Running
```

hvorefter programmet startes. Når programmet slutter, bliver kontrollen tilbagegivet til PolyPascal systemet.

Hvis der opstår en kørselsfejl under udførelsen af programmet, eller

hvis programmet afbrydes ved et tryk på Ctrl–C, bliver der udskrevet en statusmeddelelse, for eksempel:

```
EXECUTION ERROR 04 AT PC=254E
Program terminated
```

FIND kan da anvendes til at finde den sætning i kildeteksten, der var skyld i fejlen.

4.3 PROGRAM kommandoen

PROGRAM kommandoen bruges til at compilere programmet og lagre objektkoden i en diskfil. Kommandoen er afhængig af hvilken version af PolyPascal der anvendes.

PolyPascal–86
Kommandoliniens format er:

PROGRAM<filnavn>,<ssegmin>,<ssegmax>,<csegmin>,<dsegmin>

hvor <filnavn> er et filnavn og <ssegmin>, <ssegmax>, <csegmin> og <dsegmin> er hex–adresser (uden foranstillet '$' tegn). Systemet vælger automatisk '.COM' (MS–DOS) eller '.CMD' (CP/M–86) hvis filtypen udelades. Hvis <filnavn> udelades, vælges arbejdsfilens navn med typen ændret til '.COM' eller '.CMD'. Enhver af de fire hex–parametre kan udelades, for eksempel:

```
PROGRAM B:TEST,800      (kun <ssegmin>)
PROGRAM ,,,CD8,12E4     (kun <csegmin> og <dsegmin>)
```

<ssegmin> og <ssegmax> er staksegmentets minimum– og maksimumstørrelser (i paragraffer). Hvis <ssegmin> udelades, vælges 100 hex (4K bytes), og hvis <ssegmax> udelades, vælges FFFF hex (1M bytes). <csegmin> og <dsegmin> angiver minimumstørrelserne på kode– og datasegmentet (i paragraffer). Hvis intet angives, bliver den mindst mulige værdi valgt. Hverken <csegmin> eller <dsegmin> må være større end FFF hex (64K bytes). Normalt angives <csegmin> og <dsegmin> kun ved oversættelse af programmer, der "chainer" andre programmer med større kode– og/eller datasegmenter.

Når compileren startes, udskriver den:

```
Compiling to d:nnnnnnnn.ttt
```

Førend kildeteksten oversættes, skriver PolyPascal en kopi af run–time biblioteket ud i diskfilen. Run–time biblioteket optager altid den første del af en programfil.

Efter endt oversættelse udskriver systemet størrelsen af kode– og datasegmentet, samt antallet af ledige bytes (staksegmentet):

```
Code: rrrrrr bytes(aaaa paragraphs)
Data: ssssss bytes(bbbb paragraphs)
Free: tttttt bytes(cccc paragraphs)
```

De i parantes viste tal er størrelsen af de pågældende segmenter i "paragraffer". Tallene er i hexnotation, og en paragraf svarer til 16 bytes. Størrelsen af kodesegmentet omfatter både run–time biblioteket og selve programkoden.

Hvis der allerede findes en programkode i lageret når PROGRAM kommandoen udføres (på grund af en tidligere COMPILE eller RUN kommando), skriver systemet blot denne kode ud i en diskfil, i stedet for at oversætte programteksten påny.

PolyPascal–80
Kommandoliniens format er:

PROGRAM <filnavn>,<start>,<slut>

hvor <filnavn> er et filnavn og <start> og <slut> er hex–adresser (uden foranstillet '$' tegn). Systemet vælger automatisk '.COM' hvis filtypen udelades. Hvis <filnavn> udelades, vælges arbejdsfilens navn med typen ændret til '.COM'.

<start> angiver startadressen på objektkoden. Hvis <start> udelades, bliver slutadressen på run–time programdelen automatisk valgt. <start> må aldrig være mindre end slutadressen på run–time programdelen (denne adresse kan findes ved at bruge PROGRAM kommandoen uden <start> parameteren).

<slut> angiver adressen på toppen af programmets arbejdslager. Hvis <slut> udelades, bliver den nuværende adresse på toppen af arbejdslageret valgt (denne er bestemt af CP/M operativsystemet, og står i adresse 6–7). Eftersom compileren reserverer lager til variable fra toppen af arbejdslageret og nedefter, kan programmer, der er oversat for en given størrelse af arbejdslageret, ikke køres på systemer med mindre arbejdslager.

Når compileren startes, udskriver den:

```
Compiling to d:nnnnnnnn.ttt
```

Førend kildeteksten oversættes, skriver PolyPascal en kopi af run–time biblioteket ud i diskfilen. Run–time biblioteket optager altid den første del af en programfil. Området fra slutadressen på run–time programdelen til startadressen på programkoden berøres aldrig af programmet, og det er derfor et passende sted til EXTERNAL specificerede maskinkoderutiner, der for eksempel kan indsættes med DDT programmet (DDT programmet medfølger på den originale CP/M systemdisk).

Efter endt oversættelse udskriver systemet størrelsen af objektkoden og datalageret, samt antallet af ledige bytes:

```
Free: rrrrr bytes(aaaa-bbbb)
Code: sssss bytes(cccc-dddd)
Free: ttttt bytes(eeee-ffff)
Data: uuuuu bytes(gggg-hhhh)
```

De i parentes viste tal er start– og slutadresserne på det givne område (i hexnotation). Størrelsen af objektkoden omfatter ikke run–time biblioteket. Den første linie vises kun, hvis en <start> adresse blev angivet på kommandolinien.

4.4 OBJECT kommandoen

OBJECT kommandoen er magen til PROGRAM kommandoen, bortset fra, at run–time biblioteket ikke medtages i den diskfil, der genereres af compileren. Objektfiler kan kun startes ved et kald til chain proceduren fra en anden PolyPascal programfil – de kan altså ikke startes direkte fra operativsystemet. Yderligere detaljer herom findes i kapitel 19 i PolyPascal programmeringsmanualen. OBJECT kommandoen afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
Kommandoliniens format er:

```
OBJECT <filnavn>
```

hvor <filnavn> er et filnavn. Systemet vælger automatisk '.CHN' hvis filtypen udelades. Hvis <filnavn> udelades, vælges arbejdsfilens navn med typen ændret til '.CHN'.

Ved et kald til chain standardproceduren ændres størrelsen og placeringen af programmets segmenter ikke, og det er derfor ikke nødvendigt at specificere segmentparametre ved OBJECT kommandoen.

Det er imidlertid op til brugeren at specificere tilstrækkelig store segmenter, når "rod–programmet" oversættes med PROGRAM kommandoen. Størrelsen og placeringen af programsegmenterne bliver nemlig bestemt når "rod–programmet" startes fra operativsystemet. Husk derfor størrelsen af segmenterne, dvs. antallet af paragraffer for hvert segment i hver objektfil, og angiv de største af disse værdier når "rod–programmet" oversættes.

PolyPascal–80
Kommandoliniens format er:

OBJECT <filnavn>,<start>,<slut>

hvor <filnavn> er et filnavn og <start> og <slut> er hex–adresser (uden foranstillet '$' tegn). Systemet vælger automatisk '.CHN' hvis filtypen udelades. Hvis <filnavn> udelades, vælges arbejdsfilens navn med typen ændret til '.CHN'. <start> og <slut> parametrene er beskrevet under PROGRAM kommandoen i forrige afsnit.

4.5 FIND kommandoen

FIND kommandoen bruges til at finde en sætning i kildeteksten, der svarer til en given adresse i objektkoden. Når compileren startes fra en FIND kommando, genererer den ingen objektkode. Kommandoliniens format er:

FIND <offset>

hvor <offset> er den relative adresse i objektkoden på det sted, der ønskes fundet. <offset> skal angives i hex uden foranstillet '$' tegn. I PolyPascal–86 er <offset> altid den sande adresse på en instruktion i kodesegmentet. I PolyPascal–80 regnes <offset> relativt til starten af objektkoden. Hvis objektkoden for eksempel starter i adresse $1E80, vil kommandoen 'FIND 348' finde det sted i kildeteksten, der svarer til den maskinkodeinstruktion i objektkoden, der ligger på adresse $21C8.

Hvis <offset> ikke angives, vælger systemet automatisk adressen på den sidste kørselsfejl. Således kan man, blot ved at indtaste FIND efter en kørselsfejl, finde det sted i kildeteksten, der er skyld i fejlen. Førend søgningen indledes, udskriver compileren:

```
Searching
```

Hvis compileren finder den adresse, der angives, stopper den og udskriver nummeret på programlinien, for eksempel:

```
Found in line 129
Press <RETURN> to edit or <ESC> to abort
```

Ved et tryk på RETURN startes editoren, og cursoren flyttes automatisk til det sted i kildeteksten, der blev fundet. Hvis den givne adresse ikke findes, udskriver compileren:

```
Target address not found
```

hvorpå den returnerer til kommandoniveauet.

Hvis en kørselsfejl forekommer i en overlaysubrutine (en procedure eller funktion, der er lagret i en diskfil), er det ikke altid sikkert, at FIND finder det rigtige sted i kildeteksten. Dette problem er omtalt i afsnit 15.9 i PolyPascal programmeringsmanualen, og her anvises også en måde hvorpå det kan undgås.

4.6 Fejlhåndtering

Hvis compileren finder en fejl under en compilering, stopper den og udskriver et fejlnummer. Hvis der blev sagt ja ('Y') til at medtage fejlmeddelelser, da PolyPascal blev startet, bliver der desuden udskrevet en tekst, der forklarer fejlen:

```
Error 4 in line 241
Dubleret identifier
Press <RETURN> to edit or <ESC> to abort
```

Ved et tryk på RETURN startes editoren og cursoren flyttes til det fejlramte sted. Programmet kan nu rettes på sædvanlig vis.

Situationen er mere indviklet, hvis der forekommer en fejl i en include–fil. I så tilfælde udskriver compileren navnet på include–filen, samt den fejlramte linies nummer. Lad os antage, at arbejdsfilen hedder A:MAIN.PAS og at dette program inkluderer filen A:FUNCLIB.PAS. En fejlmeddelelse kan da se således ud:

```
Error 25 in file A:FUNCLIB.PAS line 156
Ukendt eller ulovlig variabelidentifier
Press <RETURN> to edit or <ESC> to abort
```

Ved et tryk på RETURN går PolyPascal i gang med at gemme teksten i lageret, og der udskrives:

```
Saving A:MAIN.PAS
```

da A:MAIN.PAS er arbejdsfilen (angivet ved brug af LOAD eller NAME kommandoen). Filen gemmes kun hvis der er blevet rettet i den, siden den blev læst eller gemt sidste gang. Derefter indlæses include–filen. I dette tilfælde vil PolyPascal udskrive:

```
Loading A:FUNCLIB.PAS
```

Tilsidst startes editoren og cursoren flyttes til det fejlramte sted. Fejlen kan nu rettes. Hvis editoren afsluttes med ^K^D kommandoen, vil PolyPascal automatisk gemme include–filen og indlæse den oprindelige arbejdsfil:

```
Saving A:FUNCLIB.PAS
Loading A:MAIN.PAS
```

Afslutter man derimod editoren med ^K^X kommandoen, bliver include–filen udnævnt til arbejdsfil.

5. Andre kommandoer

5.1 DIR kommandoen

DIR kommandoen bruges til at udskrive en disks bibliotek. Formatet af kommandolinien er:

DIR <filnavn>

hvor <filnavn> er et flertydigt filnavn, med et format magen til det, der anvendes af MS–DOS eller CP/M systemets DIR kommando, dvs. et filnavn, hvori spørgsmålstegn (?) og stjerner (*) kan indgå.

Et spørgsmålstegn angiver, at denne position kan indeholde ethvert tegn, og en stjerne angiver, at resten af tegnene i det pågældende felt (navnet eller typen) kan være hvad som helst.

Hvis både navnet og typen udelades, altså hvis kun drive–identifieren og kolonet angives, udskrives navnene på alle filer på den pågældende disk. I MS–DOS versionen gælder der desuden, at hvis <filnavn> kun består af en række biblioteksnavne, udskrives alle filer i det resulterende underbibliotek. Hvis <filnavn> udelades helt, udskrives navnene på alle filer (i det nuværende underbibliotek) på den nuværende disk.

5.2 USE kommandoen

I MS–DOS versionen bruges USE kommandoen til at vise og eventuelt ændre den nuværende disk og det nuværende underbibliotek. I CP/M versionerne bruges USE kommandoen til vise og eventuelt ændre den nuværende disk og det nuværende brugernummer, samt til at aktivere nyindsatte disketter. Kommandoliniens format afhænger af hvilken version af PolyPascal der anvendes.

MS–DOS versionen
Kommandoliniens format er:

USE <liste>

hvor <liste> er en drive–identifier og/eller en række biblioteksnavne adskilt af baglæns skråstreger. Formatet er det samme som det, der anvendes af CHDIR (CD) kommandoen i MS–DOS. Hvis <liste> udelades, udskriver USE den liste, der fører frem til det nuværende underbibliotek.

CP/M versionerne
Kommandoliniens format er:

```
USE <drive><user>
```

hvor drive er en diskidentifier (A–P) og <user> er et brugernummer (0–15). Hvis <drive> angives, bliver denne disk den nuværende disk, og hvis <user> angives, skiftes der til dette brugernummer.

USE kommandoen bruges desuden til at aktivere nyindsatte disketter. USE kommandoen skal udføres hvergang der skiftes diskette i en af diskettestationerne – ellers risikerer man, at CP/M giver en R/O fejl når man prøver at skrive på disketten.

5.3 MEMORY kommandoen

MEMORY kommandoen oplyser om størrelsen af kildeteksten og antallet af ledige bytes i arbejdslageret. Udskriften afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
PolyPascal–86 udskriver:

```
Code: rrrrrr bytes(aaaa paragraphs)
Data: ssssss bytes(bbbb paragraphs)
Free: tttttt bytes(cccc paragraphs)
```

'Code' og 'Data' linierne udskrives kun, hvis der findes en oversat version af det nuværende program i lageret. De i parantes viste tal er størrelsen af de pågældende segmenter i "paragraffer". Tallene er i hexnotation, og en paragraf svarer til 16 bytes.

PolyPascal–80
PolyPascal–80 udskriver:

```
Code: rrrrr bytes(aaaa-bbbb)
Free: sssss bytes(cccc-dddd)
```

```
Data:  t t t t t bytes(eeee-f f f f)
```

'Code' og 'Data' linierne udskrives kun, hvis der findes en oversat version af det nuværende program i lageret. Tallene i parantes er start- og slutadresserne (i hex) på hvert område.

5.4 ZAP kommandoen

ZAP kommandoen bruges til at slette teksten i lageret eller til at slette en diskfil. Kommandoliniens format er:

```
ZAP <filnavn>
```

Systemet vælger automatisk '.PAS' hvis filtypen udelades. Hvis <filnavn> angives, sletter ZAP den pågældende diskfil. Hvis <filnavn> udelades, sletter ZAP teksten i lageret og sætter arbejdsfilens navn til 'WORK.PAS'. Hvis der er blevet rettet i arbejdsfilen siden den blev indlæst eller sidst gemt, spørger systemet:

```
<arbejdsfil> not saved. Save (Y/N)?
```

Hvis der svares 'Y' bliver arbejdsfilen gemt førend den slettes fra lageret. Hvis der svares med 'N' bliver arbejdsfilen ikke gemt, og de ændringer, der eventuelt måtte være foretaget i den, går tabt. Hvis der svares med andet, afbrydes ZAP kommandoen.

5.5 QUIT kommandoen

QUIT kommandoen returnerer til operativsystemet. Hvis der er blevet rettet i arbejdsfilen siden den blev indlæst eller sidst gemt, spørger systemet:

```
<arbejdsfil> not saved. Save (Y/N)?
```

Hvis der svares 'Y' bliver arbejdsfilen gemt førend PolyPascal slutter. Hvis der svares med 'N' bliver arbejdsfilen ikke gemt, og de ændringer, der eventuelt måtte være foretaget i den, går tabt. Hvis der svares med andet, afbrydes QUIT kommandoen.

PolyPascal kan senere varmstartes som beskrevet i afsnit 1.1.

Index

A

ADJUST mode, 18, 19, 23
Arbejdsfil, 37
AUTO mode, 17, 19

B

Blokkommandoer, 20
Blokmærker, 20
Business version, 6

C

Compile,
  til arbejdslager, 27
  til objektfil, 31
  til programfil, 29
COMPILE kommandoen, 27
Compileren, 27
  fejlbehandling, 33
Cursorkommandoer, 18

D

Datasegment, 29
DIR kommandoen, 35
Distributionsdisketten, 6

E

EDIT kommandoen, 15
Editor modes, 17, 19
Editoren, 15
Editorkommandoer, 16
  afslut editering, 24
  blokke, 20
  cursorflytning, 18
  editering, 19
  hjælp, 23
  justering af linier, 23
  kontroltegn, 24
  søg/erstat, 21
  valg af tilstand, 19
Execution error, 28

F

Fejlfinding, 32
Fejlmeddelelser,
  compiler, 33
  editor, 24
  kørsel, 28
Fejltekstfilen, 6, 9
Filnavne, 7
FIND kommandoen, 32

H

HELP kommandoen, 11
Hextal, 7
Hjælpetekstfilen, 6

I

INSERT mode, 17, 19
INSTALL program, 7, 20

K

Kodesegment, 29
Kommandolinier, 10
Kommandooversigt, 10
Kontroltegn, 24
Kørselsfejl, 28
  find, 32

L

LOAD kommandoen, 13

M

MEMORY kommandoen, 36

N

NAME kommandoen, 14

O

OBJECT kommandoen, 31
  PolyPascal-80, 32
  PolyPascal-86, 31

P

PROGRAM kommandoen, 29
  PolyPascal-80, 30
  PolyPascal-86, 29

Q

QUIT kommandoen, 37

R

R/O fejl, CP/M, 36
RUN kommandoen, 28
Run-time bibliotek, 30, 31

S

SAVE kommandoen, 14
Sletning af diskfiler, 37
Sletning af teksten, 37
Staksegment, 29
Start af PolyPascal, 9
Statuslinien, 17
Systemkrav, 5
Søg/erstat kommandoer, 21

T

TAB mode, 18, 19

U

Udskiftning af diskette, 36
Underbiblioteker, 35
USE kommandoen, 35

W

WHERE kommandoen, 15

Z

ZAP kommandoen, 37

PolyPascal

Pascal Program Development System

Version 3.1

PROGRAMMERINGS-MANUAL

PolyData Microcenter A/S
Åboulevarden 13
DK–1960 København V

PolyData MicroCenter A/S
Åboulevarden 13
DK–1960 København V

Telefon: (01) 35 61 66
Telex: 27439 poly dk

Indholdsfortegnelse

0 Indledning 9

1 Sprogets grundlæggende elementer 11

1.1 Grundlæggende symboler 11
1.2 Reserverede ord og standard identifiere 12
1.3 Separatorer 14
1.4 Programlinier 14

2 Selvdefinerede sprogelementer 15

2.1 Identifiere 15
2.2 Tal 15
2.3 Strenge 16
2.4 Kommentarer 16
2.5 Compilerdirektiver 17

3 Standard skalare typer 19

3.1 Integer typen 19
3.2 Real typen 19
3.3 Boolean typen 20
3.4 Char typen 20
3.5 Byte typen 20

4 Programmer 21

4.1 Programoverskriften 21
4.2 Erklæringsdelen 21
4.2.1 Labelerklæringer 22
4.2.2 Konstantdefinitioner 22
4.2.3 Typedefinitioner 23
4.2.4 Variabelerklæringer 23
4.2.5 Procedure- og funktionserklæringer 25
4.3 Sætningsdelen 25

5 Udtryk **27**

5.1 Operatorer 27
5.1.1 Negationsoperatoren 27
5.1.2 NOT operatoren 27
5.2 Funktionskald 29

6 Sætninger **31**

6.1 Simple sætninger 31
6.1.1 Tilskrivningssætninger 31
6.1.2 Proceduresætninger 32
6.1.3 GOTO sætninger 32
6.1.4 Tomme sætninger 32
6.2 Strukturerede sætninger 32
6.2.1 Sammensatte sætninger 33
6.2.2 Betingede sætninger 33
6.2.2.1 IF sætninger 33
6.2.2.2 CASE sætninger 34
6.2.3 Repetitionssætninger 35
6.2.3.1 WHILE sætninger 35
6.2.3.2 REPEAT sætninger 35
6.2.3.3 FOR sætninger 36

7 Skalare typer og delintervaller **37**

7.1 Skalare typer 37
7.2 Delintervaller 38
7.3 Typekonvertering 39
7.4 Værdikontrol 40

8 Strenge **41**

8.1 Strengtyper 41
8.2 Strengudtryk 41
8.3 Strengtilskrivninger 42
8.4 Strengfunktioner og procedurer 42
8.4.1 fmt funktionen 45
8.5 Strenge og tegn 48
8.6 Prædefinerede strenge 49

9 Arraystrukturer **51**

9.1 Brug af arraystrukturer 51
9.2 Flerdimensionale arraystrukturer 52
9.3 Prædefinerede arraystrukturer 53
9.3.1 mem arraystrukturen 53
9.3.2 port arraystrukturen 54
9.4 Tegn–arrays 55

10 Poster **57**

10.1 Brug af poster 57
10.2 WITH sætninger 59
10.3 Variant–del i poster 60

11 Mængder **63**

11.1 Mængdetyper 64
11.2 Mængdeudtryk 64
11.2.1 Mængdeangivere 64
11.2.2 Mængdeoperatorer 65
11.3 Mængdetilskrivninger 66

12 Typeangivne konstanter **67**

12.1 Typeangivne konstanter af simple typer 67
12.2 Strukturerede konstanter 68
12.2.1 Arraykonstanter 68
12.2.2 Postkonstanter 69
12.2.3 Mængdekonstanter 70

13 Filer **71**

13.1 Filtyper 71
13.2 Operationer på filer 72
13.3 Textfiler 77
13.3.1 Operationer på textfiler 78
13.3.2 Logiske I/O enheder 80
13.3.3 Standardfiler 83
13.3.4 MS–DOS I/O omdirigering 84
13.4 Typeløse filer 87
13.4.1 Operationer på typeløse filer 87
13.5 I/O kontrol 89

14 Pointere **91**

14.1 Pointertyper 91
14.2 Brug af pointere 92
14.3 Direkte adgang til pointere 95
14.4 Oversigt over pointerrutiner 96

15 Procedurer og funktioner **99**

15.1 Parametre 99
15.2 Procedurer 100
15.2.1 Procedureerklæringer 100
15.2.2 Standardprocedurer 103
15.3 Funktioner 104
15.3.1 Funktionserklæringer 104
15.3.2 Standardfunktioner 106
15.3.2.1 Aritmetiske funktioner 106
15.3.2.2 Skalare funktioner 107
15.3.2.3 Konverteringsfunktioner 107
15.3.2.4 Andre standardfunktioner 107
15.4 FORWARD specifikationer 109
15.5 Strenge som var–parametre 110
15.6 Typeløse var–parametre 111
15.7 Absolutte procedurer og funktioner 112
15.8 Stak overflow kontrol 113
15.9 Overlay procedurer og funktioner 113
15.10 EXTERNAL specifikationer 118

16 Indlæsning og udlæsning **121**

16.1 read proceduren 121
16.2 readln proceduren 123
16.3 write proceduren 124
16.4 writeln proceduren 126

17 Afbrydning af programmer **127**

17.1 Programafbrydning under ind– og udlæsning 127
17.2 Programafbrydning under kørsel 127

18 Include–filer **129**

19 Kædning af programmer **131**

20 In–line maskinkode 135

21 Systemkald 139

21.1 PolyPascal–86 systemkald 139
21.2 PolyPascal–80 systemkald 140

22 Brugerdefinerede I/O drivere 143

23 Interne dataformater 147

23.1 Grundlæggende datatyper 147
23.1.1 Skalarer 148
23.1.2 Reals 148
23.1.3 Strenge 149
23.1.4 Mængder 149
23.1.5 Fil interface blokke 150
23.1.6 Pointere 154
23.2 Datastrukturer 154
23.2.1 Arraystrukturer 154
23.2.2 Poster 154
23.2.3 Diskfiler 155
23.2.3.1 Textfiler 155
23.2.3.2 Typeangivne filer 155
23.3 PolyPascal–86 parameteroverførsler 155
23.3.1 Parametre 156
23.3.2 Funktionsresultater 157
23.4 PolyPascal–80 parameteroverførsler 158
23.4.1 Var–parametre 158
23.4.2 Value–parametre 159
23.4.2.1 Skalarer 159
23.4.2.2 Reals 159
23.4.2.3 Strenge 159
23.4.2.4 Mængder 160
23.4.2.5 Pointere 160
23.4.2.6 Arraystrukturer og poster 160
23.4.3 Funktionsresultater 160

24 Lagerorganisering 163

24.1 PolyPascal–86 lagerorganisering 163
24.2 PolyPascal–80 lagerorganisering 164

25 Interruptstyring **169**
25.1 PolyPascal–86 interruptprocedurer 169
25.2 PolyPascal–80 interruptprocedurer 171

26 Forskelle mellem PolyPascal og Standard Pascal **175**

A Oversigt over standard procedurer og funktioner **177**

B Oversigt over operatorer **183**

C Oversigt over compilerdirektiver **185**

D ASCII tegntabel **189**

E PolyPascal syntax **191**

F I/O Fejl **201**

G Kørselsfejl **205**

H Compilerfejl **207**

Index **213**

Indledning

Formålet med denne manual er at definere PolyPascal programmeringssproget. Manualen er ikke ment som en lærebog, men den er dog så vidt som muligt ordnet således, at sprogets begreber introduceres i en logisk rækkefølge. Begyndere i programmering anbefales at supplere manualen med en lærebog i Pascal. Når man er blevet fortrolig med sproget, findes der bagerst i manualen er del appendices, der er egnede til hurtige opslag.

PolyPascal er en udvidet version af Standard Pascal, der er beskrevet af K. Jensen og N. Wirth i bogen "Pascal User Manual and Report". PolyPascal følger nøje definitionen af Standard Pascal – dog findes der enkelte forskelle, hvilket beskrives nærmere i Kapitel 26. Udvidelser, der gives i PolyPascal, omfatter blandt andet:

- Dynamiske strenge
- Random–access filbehandling
- Strukturerede konstanter
- Overlay procedurer og funktioner
- Fri ordning af elementer i erklæringsdele
- Alfanumeriske labels
- Kontroltegn i strengkonstanter
- Typekonverteringsfunktioner
- Kædning af programmer med fælles variable
- Include filer
- Fuld support af operativsystemets faciliteter
- Logiske operationer på heltal
- Bit/byte operationer
- Hexadecimale heltalskonstanter
- Direkte adgang til lager og dataporte
- Absolut adresserede variable
- In–line maskinkode

Desuden findes nye standardprocedurer og funktioner, der yderligere forbedrer sproget. Alle udvidelser er karakteriserede ved, at de enten anses for at være nødvendige for PolyPascals egnethed som et generelt programmeringssprog, eller ved, at de giver væsentlige fordele fremfor det uudvidede sprog. Udvidelserne følger i høj grad de logiske anskuelser, der ligger til grund for Standard Pascal.

PolyPascal findes i versioner til både 8086 og Z–80 microprocessorerne. I manualen kaldes 8086 versionen PolyPascal–86 og Z–80 versionen PolyPascal–80.

PolyPascal systemet og den tilhørende dokumentation er forfattet af Anders Hejlsberg og Christen Fihl.

1. Sprogets grundlæggende elementer

1.1 Grundlæggende symboler

De grundlæggende symboler i PolyPascal er inddelt i tre grupper – Bogstaver, talcifre og specialtegn:

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

Der skelnes ikke mellem store og små bogstaver. Bemærk at de danske tegn Æ, Ø, Å, æ, ø, og å ikke er klassificeret som bogstaver, og at disse derfor ikke må indgå i identifiere. Grunden hertil er, at disse tegn ikke er bogstaver i det internationale ASCII tegnsæt, men i stedet specielle tegn, hvilket illustreres af den nedenfor viste tabel:

Dansk tegn: Æ Ø Å æ ø å
ASCII tegn: [\ ] { | }

Visse operatorer og separatorer skrives som en sammensætning af to specialtegn:

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

Som vist ovenfor er det tilladt at bruge sammensatte symboler i stedet for de danske tegn – dette er bestemt at anbefale, da det gør programtekster mere overskuelige.

1.2 Reserverede ord og standard identifiere

Reserverede ord er en integreret del af PolyPascal, og de kan ikke omdefineres. Således kan de reserverede ord ikke anvendes som brugerdefinerede identifiere. De reserverede ord er:

AND	ARRAY	AT	BEGIN
CASE	CODE	CONST	DO
DOWNTO	DIV	ELSE	END
EXOR	EXTERNAL	FILE	FOR
FORWARD	FUNCTION	GOTO	IF
IN	LABEL	MOD	NIL
NOT	OF	OR	OTHERWISE
OVERLAY	PACKED	PROCEDURE	PROGRAM
RECORD	REPEAT	SET	SHL
SHR	STRING	TEXT	THEN
TO	TYPE	UNTIL	VAR
WHILE	WITH		

I denne manual skrives alle reserverede ord med store bogstaver. PolyPascal definerer også et antal standard identifiere, der ikke er reserverede, men som er navne på prædefinerede konstanter, typer, variable, procedurer og funktioner. Disse identifiere kan eventuelt overskrives med brugerens egne definitioner.

De følgende standard identifiere er fælles for alle versioner af PolyPascal:

abs	addr	allocate	alloff
arctan	assign	aux	blkoff
bklon	blockread	blockwrite	boolean
buflen	byte	chain	char
chr	close	clreol	clreos
clrhom	con	concat	copy
cos	delete	dellin	eof
eoln	erase	execute	exit
exp	false	fill	flush
frac	gotoxy	halt	hi
hptr	input	insert	inslin
int	integer	intoff	inton
iores	kbd	keypress	len
length	ln	lo	lst
mark	maxint	mem	memavail

move	new	odd	ord
output	ovclose	pi	port
pos	position	pred	ptr
pwrten	random	randomize	read
readln	real	release	rename
reset	rewrite	round	rvsoff
rvson	seek	seof	seoln
sin	size	sqr	sqrt
str	succ	swap	text
trm	true	trunc	ulnoff
ulnon	usr	val	write
writeln			

Følgende standard identifiere findes kun i MS–DOS versionen af PolyPascal–86:

aiofs	aoofs	append	chdir
ciofs	coofs	cseg	csofs
dseg	fmt	gcdir	longlen
longpos	loofs	memw	mkdir
ofs	ovpath	portw	rmdir
seg	sseg	swint	truncate
uiofs	uoofs		

Følgende standard identifiere findes kun i CP/M versionen af PolyPascal–86:

aiofs	aoofs	ciofs	coofs
cseg	csofs	dseg	fmt
loofs	memw	ofs	ovdrive
portw	seg	sseg	swint
uiofs	uoofs		

Følgende standard identifiere findes kun i PolyPascal–80:

aiaddr	aoaddr	bdos	bdosb
bios	biosb	ciaddr	coaddr
csaddr	loaddr	ovdrive	rptr
sptr	uiaddr	uoaddr	

I denne manual skrives alle standard identifiere med små bogstaver.

1.3 Separatorer

Blanktegn, linieskift og kommentarer anses for separatorer. Der skal altid være mindst en separator mellem to Pascal elementer.

1.4 Programlinier

En programlinie må højst være 127 tegn lang. Hvis en programlinie er længere end 127 tegn, bliver de overskydende tegn ignoreret.

2. Selvdefinerede sprogelementer

2.1 Identifiere

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

```
PASCAL   bredde   rod3   antal_tegn
```

Bemærk at PolyPascal ikke skelner mellem små og store bogstaver i identifiere.

2.2 Tal

Tal er konstanter af typen integer (heltal) eller real (reelle tal). Heltalskonstanter udtrykkes enten i decimal eller hexadecimal notation. Hexadecimal notation vælges ved at skrive et '$' tegn foran tallet. Det decimale heltalsområde er –32768 til 32767 og det hexadecimale talområde er $0000 til $FFFF. Eksempler på heltalskonstanter:

```
1   3741   -3   $20   $E7B9
```

Talområdet for reelle tal er 1E–38 til 1E+38 (1E–63 til 1E+63 i Business versionen, 1E–308 til 1E+308 i 8087 versionen) med en mantisse på op til 11 betydende cifre (18 i Business versionen, 15 i 8087 versionen). Eksponentiel notation vælges ved at efterfølge mantissen med et 'E', igen efterfulgt af heltallet, der udgør eksponenten. En heltalskonstant kan altid anvendes i stedet for en reel konstant. Eksempler på reelle konstanter:

```
1.0   0.025   5E10   1E-5   -3.7833564719E+12
```

Talkonstanter må ikke indeholde separatorer.

2.3 Strenge

En tekststreng er en følge af tegn omsluttet af enkelte anførelsestegn. Anførelsestegnet kan indgå i strengen, hvis det skrives dobbelt. En streng, der indeholder et enkelt tegn, er en konstant af typen char. En streng, der indeholder flere end et tegn, er en konstant af typen ARRAY[1..n] OF char. Strenge, uanset deres længde, er altid kompatible med alle STRING typer. Eksempler på strenge:

```
'PolyPascal'  'That''s all folks'  ';'  ''''  ''
```

Bemærk det sidste eksempel – anførelsestegnene omslutter ingen (nul) tegn og angiver dermed den tomme streng.

I PolyPascal er det også muligt at lade kontroltegn indgå i strengkonstanter. Der findes to notationer for kontroltegn: Et snabel-a (@) efterfulgt af en heltalskonstant angiver et tegn af denne ASCII værdi, og en pil (^) efterfulgt af et tegn, med ASCII værdi n, angiver kontroltegnet med ASCII værdien n–64. Nogle eksempler på kontroltegnsangivelser:

```
@13  @127  @$1C  ^M  ^G  ^L  ^Z
```

Flere kontroltegn kan sammensættes til strenge ved at skrive dem umiddelbart efter hinanden:

```
@13@10  @27@61@32@32  ^M^J  ^G^G^G  ^Z@31@31^J
```

De ovenfor viste strenge indeholder henholdsvis to, fire, to, tre, og fire tegn. Kontroltegn og tekststrenge kan også sammensættes:

```
'FEJL '^G^G^G' PRØV IGEN'  @27'[1;1H'  'Linie 1'^M^J
```

De her viste strenge indeholder henholdsvis atten, seks, og ni tegn.

2.4 Kommentarer

En kommentar kan indsættes hvorsomhelst i et program mellem to sprogelementer. Kommentarer skal omsluttes af symbolerne { og } eller af symbolerne (* og *). Et eksempel:

```
(* dette er en kommentar *)  {dette er også en kommentar}
```

2.5 Compilerdirektiver

Visse af PolyPascal compilerens faciliteter styres ved hjælp af compilerdirektiver. Et compilerdirektiv er faktisk et specialtilfælde af en kommentar, og kan derfor anvendes nårsomhelst en kommentar er tilladt. En liste af compilerdirektiver startes med et $ tegn umiddelbart efter den indledende kommentarparentes. Selve listens syntaks afhænger af, hvilke direktiver der angives. En fuld beskrivelse af hvert compilerdirektiv følger senere i manualen, og en oversigt findes i Appendix C. Nogle eksempler på compilerdirektiver:

```
{$I-}  {$A+,R-,B+}  {$I STRINP.LIB}  (*$W5*)
```

Bemærk at der ikke må være blanktegn i {$ eller (*$ symbolerne eller umiddelbart efter $ tegnet.

3. Standard skalare typer

En datatype er afgørende for, hvilke værdier en variabel kan antage. I et program skal enhver variabel være forbundet med een og kun een datatype. Blandt datatyperne findes visse grundlæggende typer, som normalt kaldes standard skalare datatyper. Disse er integer, real, boolean, char, og byte.

3.1 Integer typen

En integer er et heltal i området –32768 til 32767, eller i området $0000 til $FFFF. Variable af typen integer optager to bytes i datalageret.

Overløb ved heltalsoperationer rapporteres ikke. Alle mellemresultater i et heltalsudtryk skal holdes inden for det lovlige heltalsområde, da resultatet ellers bliver ukorrekt. For eksempel vil udtrykket 4000*50 DIV 25 ikke give 8000, da multiplikationen resulterer i et overløb.

3.2 Real typen

I PolyPascal–80 og i standardversionen af PolyPascal–86 er talområdet for reelle tal 1E–38 til 1E+38 med 11 betydende cifre, og variable af typen real optager 6 bytes. I Business versionen af PolyPascal–86 er talområdet 1E–63 til 1E+63 med 18 betydende cifre, og variable af typen real optager 10 bytes. I 8087 versionen af PolyPascal–86 er talområdet 1E–308 til 1E+308 med 15 betydende cifre, og variable af typen real optager 8 bytes.

I tilfælde af overløb i et aritmetisk udtryk af typen real, stopper programmet og udskriver en fejlmeddelelse. I tilfælde af underløb returnerer operationen nul (0.0).

Selvom real typen er en skalar datatype, kan den ikke altid bruges i de samme tilfælde som andre skalare typer: Standardfunktionerne pred og succ tillader ikke argumenter af typen real, indextypen i et array må ikke være real, grundtypen i en mængde må ikke være real, kontrolvariablen i en FOR sætning må ikke være af typen real, nøgleudtrykket i en CASE sætning må ikke være af typen real, og et delinterval med grundtype real er ikke tilladt.

3.3 Boolean typen

Boolean variable kan antage to værdier, sand eller falsk, givet ved standardkonstanterne true og false. Boolean typen er defineret således at false < true. En boolean variabel optager en byte i datalageret.

3.4 Char typen

En char værdi er et tegn i ASCII tegnsættet. Tegn er ordnet efter deres ASCII værdi – således gælder der, for eksempel, at 'A' < 'B'. Den ordinale værdi (ASCII værdien) af et tegn må være mellem 0 og 255, dvs. fra @0 til @255. En char variabel optager en byte i datalageret.

3.5 Byte typen

Typen byte er faktisk et delinterval af typen integer, defineret som TYPE byte = 0..255. Således er byte variable kompatible med integer variable, dvs. bytes og integers kan blandes i udtryk, og bytes kan tilskrives værdier af typen integer. En variabel af typen byte optager en byte i datalageret.

4. Programmer

Ethvert program består af en programoverskrift efterfulgt af en programblok. Programblokken indeholder en erklæringsdel, hvori programmets elementer (labels, konstanter, typer, variable, procedurer og funktioner) erklæres, og en sætningsdel, hvori programmets handlinger opskrives i form af programsætninger.

4.1 Programoverskriften

Programoverskriften tildeler programmet et navn, og angiver eventuelt identifiere for en eller flere af de I/O kanaler, programmet bruger. Listen af identifiere for I/O kanaler er omsluttet af parenteser, og hver enkelt identifier er adskilt fra de omkringstående af kommaer. Eksempler på programoverskrifter:

```
PROGRAM kvadrater;
PROGRAM lommeregner(input,output);
PROGRAM sortering(printer,disk);
```

I PolyPascal har programoverskriften ingen reel betydning for programmet, og den kan derfor udelades efter behag.

4.2 Erklæringsdelen

I programblokkens erklæringsdel erklæres de identifiere, der bruges i blokken og andre blokke inden i denne. Erklæringsdelen er underdelt i fem forskellige afsnit:

1. Labelerklæringer
2. Konstantdefinitioner
3. Typedefinitioner
4. Variabelerklæringer
5. Procedure– og funktionserklæringer

I PolyPascal må hvert afsnit skrives et vilkårligt antal gange, og afsnittenes rækkefølge er uden betydning. Standard Pascal angiver dog, at hvert afsnit kun må forekomme nul eller en gang, og kun i den ovennævnte rækkefølge.

4.2.1 Labelerklæringer

Enhver sætning i et program kan mærkes med en label ved at skrive labelens identifier efterfulgt af et kolon foran sætningen. Det kræves imidlertid, at sådanne labels er erklæret i en labelerklæringsdel, før de bruges. En labelerklæringsdel indledes med det reserverede ord LABEL, efterfulgt af en liste af labelidentifiere, adskilt af kommaer, og afsluttes med et semikolon. Et eksempel:

```
LABEL 100,error,999,stop;
```

Bemærk at Standard Pascal foreskriver, at labels kun må være positive heltal mellem 0 og 9999, medens PolyPascal tillader både tal og identifiere.

4.2.2 Konstantdefinitioner

En konstantdefiniton erklærer en identifier, og tildeler den en fast værdi. En konstantdefinitionsdel indledes med det reserverede ord CONST, efterfulgt af en række konstanttilskrivninger, adskilt af semikoloner. Hver konstanttilskrivning består af en identifier efterfulgt af et lighedstegn og en konstant. Konstanter er enten tal eller strenge, som beskrevet i afsnit 2.2 og 2.3. Et eksempel:

```
CONST
  antal = 45;
  max = 193.158;
  min = -max;
  navn = 'Michael';
  nylinie = ^M^J;
```

De følgende konstanter er prædefinerede i PolyPascal:

pi	real	3.1415926536E+00.
false	boolean	Sandhedsværdien "falsk".
true	boolean	Sandhedsværdien "sand".
maxint	integer	32767.

En konstantdefinitionsdel kan også definere typeangivne konstanter. Dette beskrives i kapitel 12.

4.2.3 Typedefinitioner

En datatype i Pascal kan enten beskrives direkte i en variabelerklæring, eller den kan referes til via en typeidentifier. En del prædefinerede typeidentifiere findes som standard (integer, real, boolean, etc.), men udover disse kan brugeren definere nye typer i en typedefinitionsdel. En sådan indledes med det reserverede ord TYPE, efterfulgt af en række typetilskrivninger, adskilt af semikoloner. Hver typetilskrivning består af en typeidentifier efterfulgt af et lighedstegn og en datatype. Et eksempel:

```
TYPE
  heltal = integer;
  byte = 0..255;
  dag = (man,tir,ons,tor,fre,lor,son);
  liste = ARRAY[1..10] OF real;
  complex = RECORD re,im: real END;
```

Yderligere eksempler på typedefinitioner findes i de følgende kapitler.

4.2.4 Variabelerklæringer

Alle variable, der anvendes i et program, skal erklæres i variabelerklæringsdelen. Erklæring af en variabel skal altid gå forud for dens brug, eller, med andre ord, variablen skal være "kendt" af compileren, førend man kan referere til den.

En variabelerklæring erklærer en ny variabelidentifier og forbinder den med en datatype. Denne sammenhæng mellem identifier og type gælder i hele den blok, hvori variabelerklæringen foretages, med mindre identifieren redefineres i en indre blok. Variabelerklæringsdelen indledes med det reserverede ord VAR, efterfulgt af et antal variabelerklæringer, adskilt af semikoloner. Hver variabelerklæring består af en eller flere identifiere, adskilt af kommaer, efterfulgt af et kolon og en datatype. Et eksempel:

```
VAR
  rod1,rod2,rod3: real;
  taeller,i: integer;
  fundet: boolean;
  d1,d2: dag;
  buffer: ARRAY[0..127] OF byte;
```

Variable kan eventuelt erklæres til at ligge på faste adresser i lageret. Dette gøres ved at tilføje en AT specifikation til variabelerklæringen efter datatypen. Syntaksen for en AT specifikation afhænger af hvilken version af PolyPascal der anvendes:

PolyPascal–86
Det reserverede ord AT skal efterfølges af to heltalskonstanter, adskilt af et kolon. Disse angiver segment– og offsetadresserne på den første byte i lageret, der skal optages af variablen. Et eksempel:

```
VAR
  int_3_ofs: integer AT $0000:$000C;
  int_3_seg: integer AT $0000:$000E;
```

En variabel kan placeres på en absolut offset–adresse i kode– eller datasegmentet ved at bruge identifieren cseg eller identifieren dseg efterfulgt af et kolon og en offsetadresse:

```
VAR
  kommandolinie: STRING[127] AT dseg:$80;
```

PolyPascal–80
Det reserverede ord AT skal efterfølges af en heltalskonstant, der angiver adressen på den første byte i lageret, der skal optages af variablen. Et eksempel:

```
VAR
  memtop: integer AT $0006;
  kommandolinie: STRING[127] AT $0080;
```

AT specifikationen kan også angive, at en variabel skal ligge "oven i" en anden variabel, dvs. at den nye variabel skal starte på den samme adresse som den variabel, der angives efter AT. Et eksempel:

```
VAR
  str: STRING[32];
  laengde: byte AT str;
```

Med de ovenfor viste erklæringer kommer str og laengde til at starte på samme adresse i lageret (da den første byte af en strengvariabel indeholder længden af strengen, vil variablen laengde altså indeholde længden af strengen str).

Der må ikke erklæres mere end en variabel ad gangen, når AT specifikationen anvendes.

Kapitel 23 og 24 giver yderligere detaljer om allokering af lager til variable.

4.2.5 Procedure– og funktionserklæringer

Procedureerklæringsdelen bruges til at erklære underprogrammer i den nuværende programblok (se afsnit 15.2). En procedure aktiveres via af en proceduresætning (se afsnit 6.1.2).

Funktionserklæringsdelen bruges til at erklære underprogrammer, der udregner og returnerer en værdi (se afsnit 15.3). En funktion aktiveres via et funktionskald, der indgår i et udtryk (se afsnit 5.2).

4.3 Sætningsdelen

Sætningsdelen er den sidste del af en programblok. Den angiver de handlinger, der skal udføres når programmet eksekveres. Sætningsdelen svarer til en sammensat sætning efterfulgt af et punktum. En sammensat sætning består af et vilkårligt antal sætninger, adskilt af semikoloner, og omsluttet af de reserverede ord BEGIN og END.

Et eksempel på et program:

```
PROGRAM konverter(output);
CONST
  offset = 32; faktor = 1.8;
  start = 10; slut = 19;
  separator = '----------';
TYPE
  gradtype = start..slut;
VAR
  grad: gradtype;
BEGIN
  writeln(separator);
  FOR grad:=start TO slut DO
  BEGIN
    write(grad:10,'c',round(grad*faktor+offset):10,'f');
    IF odd(grad) THEN writeln;
  END;
  writeln(separator);
END;
```

Programmet giver den følgende udskrift:

10c	50f	11c	52f
12c	54f	13c	55f
14c	57f	15c	59f
16c	61f	17c	63f
18c	64f	19c	66f

5. Udtryk

Udtryk består af operander og operatorer. Operander kan være variable, konstanter og funktionskald. Operatorerne er inddelt i fem prioritetsniveauer. Negationsoperatoren (et minus med en operand) har den højeste prioritet, efterfulgt af NOT operatoren, igen efterfulgt af de multiplicerende operatorer, derefter de adderende operatorer, og til sidst de relationelle operatorer. En følge af operatorer med samme prioritet evalueres fra venstre mod højre. Udtryk omsluttet af parenteser evalueres uafhængigt af de omkringstående operatorer.

Dette kapitel beskriver udtryk af de standard skalare datatyper, dvs. integer, real, boolean og char. Udtryk af selvdefinerede skalare typer, strengtyper og mængdetyper kan også konstrueres, hvilket beskrives nærmere i afsnittene 7.1, 8.2 og 11.2.

5.1 Operatorer

Hvis begge operander i en addition (+), subtraktion (–) eller multiplikation (*) er af typen integer, er resultatet også af typen integer. Hvis en eller begge operander er af typen real, er resultatet ligeledes af typen real.

5.1.1 Negationsoperatoren

Negationsoperatoren (et minus med en operand) indikerer, at operandens fortegn skal inverteres. Operanden kan være af typen real eller af typen integer.

5.1.2 NOT operatoren

NOT operatoren angiver at operanden, der er af typen boolean, skal komplementeres:

```
NOT true    = false
NOT false   = true
```

PolyPascal tillader også, at NOT operatoren anvendes på en operand af typen integer. I så tilfælde angiver NOT, at samtlige 16 bits i operanden skal inverteres. Nogle eksempler:

```
NOT 0       = -1
NOT -7      = 6
NOT $23A5   = $DC5A
```

5.1.3 Multiplicerende operatorer

Operator	Operation	Operandtyper	Resultattype
*	Multiplikation	real, integer	real, integer
/	Division	real, integer	real
DIV	Heltalsdivision	integer	som operand
MOD	Modulus	integer	som operand
AND	Logisk AND	integer, boolean	som operand
SHL	Venstreskift	integer	som operand
SHR	Højreskift	integer	som operand

5.1.4 Adderende operatorer

Operator	Operation	Operandtyper	Resultattype
+	Addition	real, integer	real, integer
-	Subtraktion	real, integer	real, integer
OR	Logisk OR	integer, boolean	som operand
EXOR	Logisk EXOR	integer, boolean	som operand

5.1.5 Relationelle operatorer

De relationelle operatorer tillader operander af alle standard skalare typer, dvs. integer, real, boolean og char. Integer og real operander kan eventuelt blandes. Resultattypen er altid boolean, dvs. resultatet er enten true eller false.

a = b	Sandt hvis a er lig med b.
a <> b	Sandt hvis a er forskellig fra b.
a > b	Sandt hvis a er større end b.
a < b	Sandt hvis a er mindre end b.
a >= b	Sandt hvis a er større end eller lig med b.
a <= b	Sandt hvis a er mindre end eller lig med b.

5.2 Funktionskald

Et funktionskald til en standardfunktion eller en selvdefineret funktion kan indgå i et udtryk ved at angive funktionens identifier, eventuelt efterfulgt af en parameterliste. En parameterliste er en følge af variable eller udtryk, adskilt af kommaer, og omsluttet af parenteser. Funktionens resultat kan betragtes som en variabel af samme datatype som funktionen. Eksempler:

```
round(grad)
sqrt(sqr(x)+sqr(y))
(max(a,b)<10) AND (c>100)
volumen(radius,hoejde)
```

6. Sætninger

Sætningsdelen i et program, en procedure eller en funktion består af en sammensat sætning, dvs. en følge af sætninger, adskilt af semikoloner og omsluttet af de reserverede ord BEGIN og END. Disse sætninger angiver de handlinger, der skal udføres, når programmet, proceduren eller funktionen udføres.

Enhver sætning kan forudgås af en eller flere labels, der kan referes i en GOTO sætning (se afsnit 4.2.1 og 6.1.3).

Sætninger i Pascal er inddelt i simple sætninger og strukturerede sætninger.

6.1 Simple sætninger

En simpel sætning er en konstruktion, der ikke indeholder andre sætninger. I denne gruppe findes tilskrivningssætninger, proceduresætninger, GOTO sætninger og tomme sætninger.

6.1.1 Tilskrivningssætninger

Tilskrivningssætningen er den mest fundamentale af alle sætninger. Den angiver, at en variabel skal tilskrives en ny værdi, der udregnes i et udtryk. Sætningen består af en variabel efterfulgt af tilskrivningsoperatoren (:=) og et udtryk.

Tilskrivninger kan foretages til variable af alle datatyper, dog ikke til filvariable. Bemærk, at variablen og udtrykket skal være af samme type, med den undtagelse, at hvis variabeltypen er real, må udtrykkets type gerne være integer. Eksempler på tilskrivningssætninger:

```
antal:=antal+1;
grad:=grad-10;
fundet:=false;
afstand:=sqrt(sqr(x)+sqr(y));
ciffer:=(num>=0) AND (num<=9);
rod:=(-b+sqrt(sqr(b)-d))/(2*a);
a:=max3(a,b,c);
```

6.1.2 Proceduresætninger

En proceduresætning bruges til at aktivere en standardprocedure eller en selvdefineret procedure. Sætningen består af en procedureidentifier, eventuelt efterfulgt af en parameterliste. Parameterlisten er en følge af variable eller udtryk, adskilt af kommaer og omsluttet af parenteser. Eksempler på proceduresætninger:

```
seek(f,r)
sorter(navne)
ombyt(x,y)
plot(x,round(sin(x*f)*20.0)+24)
```

6.1.3 GOTO sætninger

En GOTO sætning indikerer, at programudførslen skal fortsættes fra den label, der referes til. Ved brug af GOTO sætninger bør de følgende regler iagttages:

En label kan kun refereres til inden for den blok, hvori den er erklæret. Det er således ikke tilladt at hoppe ind og ud af procedureblokke.

Enhver label skal erklæres i en labelerklæring i den blok, hvori den bruges.

6.1.4 Tomme sætninger

En tom sætning indeholder ingen symboler og angiver således heller ingen handling. En tom sætning forekommer de steder, hvor Pascals syntaks forventer en sætning, men hvor der ingen er. Eksempler:

```
BEGIN END;
WHILE digit AND (a>17) DO {ingenting};
REPEAT {vent} UNTIL keypress;
```

6.2 Strukturerede sætninger

En struktureret sætning er en konstruktion, der bl.a. indeholder andre sætninger. Disse udføres enten sekventielt (sammensat sætning), betinget (betingede sætninger) eller repeterende (repetitionssætninger).

6.2.1 Sammensatte sætninger

I nogle tilfælde må man kun anvende en enkelt sætning som komponent i en struktureret sætning. Såfremt man i denne situation ønsker at skrive flere sætninger, kan man bruge en sammensat sætning, der er en følge af sætninger, adskilt af semikoloner og omsluttet af de reserverede ord BEGIN og END. Et eksempel:

```
IF x<y THEN
BEGIN
  temp:=x; x:=y; y:=temp;   (* ombyt x og y *)
END;
```

Det er ikke nødvendigt at skrive et semikolon efter den sidste sætning i en sammensat sætning.

6.2.2 Betingede sætninger

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

6.2.2.1 IF sætninger

IF sætningen angiver, at den indgående sætning kun skal udføres, hvis et givet boolean udtryk er sand (true). Hvis udtrykket er falsk (false), gælder der enten, at der ikke udføres nogen handling, eller, at sætningen efter ELSE symbolet skal udføres. Konstruktionen:

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

skal forstås på følgende måde:

Hvis <e1> er falsk, udføres der ingen handling.
Hvis <e1> er sand, og <e2> er sand, udøres <s1>.
Hvis <s1> 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 på IF sætninger:

```
IF x<1.5 THEN z:=x+y ELSE z:=1.5;

IF tal<0 THEN
BEGIN
  writeln('Negative tal tillades ikke');
  tal:=0;
END;
```

6.2.2.2 CASE sætninger

En CASE sætning består af et udtryk (kaldet nøgleudtrykket) og en liste af sætninger, hver foregået af en værdiliste. En sætning bliver udført, hvis værdien af valgudtrykket findes i sætningens værdiliste. Hvis nøgleudtrykkets værdi ikke findes i nogen af værdilisterne, gælder der enten, at ingen sætninger udføres, eller, at sætningerne mellem de reserverede ord OTHERWISE og END udføres.

En værdiliste består af et vilkårligt antal konstanter eller intervaller, adskilt med kommaer, efterfulgt af et kolon. Et interval skrives som to konstanter adskilt af et "dovent kolon" (..). Typen af konstanterne skal være den samme som typen af nøgleudtrykket. Sætningen efter værdilisten udføres, hvis nøgleudtrykkets værdi er lig med en af konstanterne, eller indeholdt i et af intervallerne.

Nøgleudtrykket kan være af enhver skalar type, undtagen real. Nogle eksempler:

```
CASE operator OF
  '+': x:=x+y;
  '-': x:=x-y;
  '*': x:=x*y;
  '/': x:=x/y;
END;

CASE tal OF
  1,3,5,7,9: writeln('ulige ciffer');
  2,4,6,8: writeln('lige ciffer');
  0,10..255: writeln('nul eller mellem 10 og 255');
```

```
OTHERWISE
  writeln('negativt eller større end 255');
  taeller:=taeller+1;
END;
```

Den sidste sætning før det reserverede ord OTHERWISE og den sidste sætning før det reserverede ord END behøver ikke efterfølges af et semikolon.

6.2.3 Repetitionssætninger

En repetitionssætning bruges, når man ønsker at udføre en eller flere sætninger gentagne gange. Er antallet af gentagelser på forhånd kendt, bør FOR sætningen bruges. Ellers kan WHILE og REPEAT sætningerne bruges.

6.2.3.1 WHILE sætninger

Udtrykket, der kontrollerer WHILE sætningen, skal være af typen boolean. Den indskrevne sætning gentages sålænge udtrykket er sand (true). Hvis udtrykket er falsk i begyndelsen, bliver sætningen aldrig udført. Nogle eksempler:

```
WHILE tal<1000 DO tal:=sqr(tal);

WHILE i>0 DO
BEGIN
  IF odd(i) THEN z:=z*x;
  i:=i DIV 2;
  x:=sqr(x);
END;
```

6.2.3.2 REPEAT sætninger

Udtrykket, der kontrollerer REPEAT sætningen, skal være af typen boolean. Følgen af sætninger mellem de reserverede ord REPEAT og UNTIL gentages, indtil udtrykket bliver sand (true). Et eksempel:

```
REPEAT
  readln(tal); sum:=sum+tal;
UNTIL tal=0;
```

Det er ikke nødvendigt at skrive et semikolon efter den sidste sætning i en REPEAT sætning.

6.2.3.3 FOR sætninger

FOR sætningen angiver, at den indgående sætning skal udføres gentagne gange, mens en stigende eller faldende række værdier bliver tilskrevet en variabel, kaldet kontrolvariablen. Værdierne kan enten stige i spring af 1 (TO) eller falde i spring af 1 (DOWNTO).

Kontrolvariablen, startværdien og slutværdien skal være af samme skalare type. Typen real er ikke tilladt. Kontrolvariablen skal være en enkeltvariabel (dvs. den må ikke være et element i en struktur).

Hvis startværdien er større end slutværdien ved TO, eller hvis startværdien er mindre end slutværdien ved DOWNTO, udføres den indgående sætning ikke. Nogle eksempler:

```
FOR i:=1 TO 10 DO writeln(i:5,sqr(i):5);

FOR i:=1 TO n DO
BEGIN
  readln(tal);
  IF tal=0 THEN
  antalnuller:=antalnuller+1 ELSE
  IF tal>0 THEN
  positivsum:=positivsum+tal ELSE
  negativsum:=negativsumtal;
END;
```

Bemærk, at den indgående sætning ikke må udføre tilskrivninger til kontrolvariablen. Hvis FOR løkken skal afbrydes, før den slutter af sig selv, bør en GOTO sætning anvendes (dette er imidlertid dårlig programmeringsteknik – anvend i stedet en WHILE eller en REPEAT sætning).

Efter endt udførsel af en FOR sætning er kontrolvariablen lig med slutværdien, med mindre FOR sætningen blev oversprunget. I sidstnævnte tilfælde bliver der ikke foretaget tilskrivning til kontrolvariablen.

7. Skalare typer og delintervaller

Skalare typer er de grundlæggende datatyper i Pascal. Det, der kendetegner disse typer, er, at de indeholder en endelig og lineært ordnet mængde af værdier.

Datatypen real regnes for en skalar type selvom den ikke opfylder den ovenfor givne definition. Af denne grund kan reals ikke altid anvendes i de sammenhænge, hvor andre skalare typer tillades.

7.1 Skalare typer

Udover at benytte de standard skalare typer (integer, real, boolean og char) kan brugeren selv definere nye skalare typer. Dette gøres ved at angive en liste af identifiere, der er de mulige værdier for den nye type, i den rækkefølge, der skal gælde for værdierne. Nogle eksempler:

```
TYPE
  kort = (kloer,ruder,hjerter,spar);
  dag = (man,tirs,ons,tors,fre,loer,soen);
  operator = (plus,minus,gange,dividere);
  tipstegn = (et,kryds,to);
```

Variable af den ovenstående type kort kan antage en af fire værdier, nemlig kloer, ruder, hjerter eller spar. Den standard skalare type boolean er defineret som:

```
TYPE
  boolean = (false,true);
```

De relationelle operatorer (=, <>, >, <, >= og <=) kan bruges på alle selvdefinerede skalare typer, forudsat at begge operander er af den samme type (reals og integers kan dog blandes). Den definerede rækkefølge, dvs. den rækkefølge hvori typens konstanter er angivet i typedefinitionen, lægges til grund for sammenligningen. For den ovenstående type kort, gælder der således:

```
kloer < ruder < hjerter < spar
```

Der findes tre standardfunktioner, der kan anvendes på alle skalare typer (undtagen real):

succ(x)	returnerer efterfølgeren til x.
pred(x)	returnerer forgængeren til x.
ord(x)	returnerer ordinalværdien af x.

Resultattypen for succ og pred er den samme som argumentets type. Resultattypen for ord er integer. Den ordinale værdi af den første konstant i en selvdefineret skalar type er 0. For de ovenstående skalare typer gælder der:

```
succ(ruder) = hjerter
pred(fre) = tors
ord(gange) = 2
```

7.2 Delintervaller

Man kan definere en ny datatype, som er et delinterval (engelsk "subrange") af en tidligere defineret skalar type (undtagen real). Definitionen angiver blot den mindste og den største værdi i delintervallet, hvor den nedre grænse skal være mindre end eller lig med den øvre. Delintervaller af typen real tillades ikke. Nogle eksempler:

```
TYPE
  dag = (man,tirs,ons,tors,fre,loer,soen);
  arbejdsdag = man..fre;
  fridag = loer..soen;
  ciffer = '0'..'9';
  bogstav = 'A'..'Z';
  skala = -99..99;
  maanedlaengde = 28..31;
```

De ovenstående typer arbejdsdag og fridag er begge delintervaller af den skalare type dag, også kendt som deres overordnede type. Den overordnede type for ciffer og bogstav er char, og den overordnede type for skala og maanedlaengde er integer. Den prædefinerede type byte er givet ved:

```
TYPE
  byte = 0..255;
```

En delintervaltype arver alle de egenskaber, der kendetegner dens overordnede skalare type, blot med en begrænsning af værdiområdet.

Brug af skalare typer og delintervaller kan anbefales af mange grunde. For det første letter de i høj grad forståelsen af et program. For det andet bliver værdier, der tilskrives variable af skalare typer og delintervaltyper, automatisk kontrolleret, når et program udføres (med mindre andet angives), og for det tredje vil skalarer og delintervaller ofte spare lagerplads, da PolyPascal compileren kun reserverer en byte til variable af selvdefinerede skalare typer med op til 256 elementer, og variable af delintervaltyper, hvor både øvre og nedre grænse er mellem 0 og 255.

7.3 Typekonvertering

Som tidligere beskrevet i dette afsnit kan ord funktionen anvendes til at konvertere værdier af skalare typer til værdier af typen integer. Standard Pascal definerer imidlertid ikke en metode til den omvendte situation, dvs. en metode hvormed integers kan konverteres til skalarer.

I PolyPascal kan en værdi af enhver skalar type konverteres til en værdi af enhver anden skalar type, med den samme ordinale værdi, ved hjælp af "retype" faciliteten. Dette opnås ved at anvende typeidentifieren for den ønskede skalare type i et funktionskald. Funktionskaldet skal angive en enkelt parameter, omsluttet af parenteser, der må være en værdi af enhver skalar type. Under forudsætning af typedefinitionerne i afsnit 7.1 gælder der således:

```
integer(hjerter) = 2
dag(4) = fre
operator(0) = plus
char(65) = 'A'
integer('0') = 48
operator(ruder) = minus
```

Typekonverteringer må ikke anvende typen real, hverken som den ønskede type eller som argumenttypen.

7.4 Værdikontrol

Medmindre andet angives, genererer PolyPascal compileren kode i det oversatte program til at foretage kontrol af værdier, der tilskrives variable af selvdefinerede skalare typer og delintervaltyper. Dette er ofte en stor fordel når et program skal fejlsøges, men i et færdigt program er det sjældent nødvendigt. Ved hjælp af {$R+} og {$R–} compilerdirektiverne er det derfor muligt at styre, om compileren skal generere kode til at foretage kontrol. Ved start af compileren bliver {$R+} automatisk valgt, og når variable af skalare typer og delintervaltyper tilskrives i denne stilling, vil de tilskrevne værdier blive kontrolleret. I den modsatte stilling, {$R–}, bliver der ikke foretaget kontrol. Et eksempel:

```
PROGRAM vaerdikontrol;
TYPE
  ciffer = 0..9;
VAR
  c1,c2,c3: ciffer;
BEGIN
  c1:=5;                {tilladt}
  c2:=c1+1;             {tilladt da c1<9}
  {$R-} c3:=167;        {ulovligt men giver ikke fejl}
  {$R+} c3:=65;         {ulovligt og giver kørselsfejl}
END.
```

Når værdikontrol er aktiveret bliver der også genereret kode til at checke parameterværdier i procedure– og funktionskald, hvis parameteren er en værdiparameter af en skalar type eller en delintervaltype.

Værdikontrol bør kun slås fra i gennemprøvede programmer.

8. Strenge

Strenge, der er følger af tegn, bruges ofte i programmer. Til strengbehandling i PolyPascal er der mulighed for at definere strengtyper. Antallet af tegn i en strengvariabel, også kaldt længden af strengen, kan variere dynamisk mellem 0 og en given øvre grænse.

8.1 Strengtyper

Når en strengtype defineres, angiver man den maksimale længde af de strengværdier, der kan tilskrives strengvariable af denne type. Strengtypen indledes med det reserverede ord STRING, efterfulgt af en konstant der angiver den maksimale længde. Konstanten skal være et heltal mellem 1 og 255, omsluttet af [og] symbolerne. Nogle eksempler:

```
TYPE
  filnavn = STRING[14];
  linie = STRING[72];
  hexstr = STRING[4];
```

Antallet af bytes, der optages i datalageret af en variabel af en given strengtype, svarer til strengvariablens maksimale længde plus en.

8.2 Strengudtryk

Strengværdier kan udregnes fra andre strengværdier ved hjælp af strengudtryk. I lighed med numeriske udtryk er strengudtryk opbygget af operander og operatorer. Operanderne er enten strengkonstanter, strengvariable eller strengfunktioner.

De relationelle operatorer (=, <>, >, <, >= og <=) kan bruges til at sammenligne strenge. Ved en sådan sammenligning bliver tegnene i strengene sammenlignet enkeltvis fra venstre mod højre, indtil to tegn er forskellige. Hvis strengene er af forskellig længde, men ens til længden af den korteste streng, anses den korteste streng for at være den mindste. To strenge er ens, hvis og kun hvis de er af samme længde og indeholder de samme tegn i den samme rækkefølge. Nogle eksempler på sammenligninger, der giver værdien true:

```
'streng' = 'streng'
'B' > 'A'
'ABC' < 'ABCD'
'test ' <> 'test'
'12' < '2'
```

Sammensætningsoperatoren, der skrives som et plus tegn (+), bruges til at sammensætte strenge. Sammensætningsoperatoren har højere prioritet end de relationelle operatorer. Nogle eksempler:

```
'Jens '+'Hansen' = 'Jens Hansen'
'132'+'.'+'377' = '132.377'
'A'+'B'+'C'+'D' = 'ABCD'
```

8.3 Strengtilskrivninger

Tilskrivningsoperatoren (:=) bruges til at tilskrive strengværdier til strengvariable. Nogle eksempler:

```
cifre:='0123456789';
linie:='dette er en streng';
tekst:=''+linie+'';
```

Hvis den streng, der tilskrives en strengvariabel, er længere end strengvariablens maksimale længde, er det kun de første tegn, der overføres. Hvis, for eksempel, strengen 'langstreng' tilskrives en strengvariabel af typen STRING[4], vil variablen kun indeholde de første fire tegn, dvs. 'lang'.

8.4 Strengfunktioner og procedurer

De følgende strengfunktioner er standard i PolyPascal:

len(s)	Returnerer længden af strengudtrykket s, dvs. antallet af tegn i s. Resultattypen er integer.

pos(p,s) p og s er strengudtryk, og resultattypen er integer. Funktionen returnerer positionen af den første forekomst af strengen p i strengen s (positionen af det første tegn i en streng er 1). Hvis p ikke findes i s, returneres 0.

copy(s,i,n) s er et strengudtryk, og i og n er udtryk af typen integer. copy returnerer en streng, der indeholder n tegn fra s, startende fra den i'te position i s. Hvis i er større end len(s), returneres en tom streng. Hvis i+n–1 er større end len(s), returneres en streng på len(s)–i+1 tegn. Hvis i er udenfor området 1..255, stopper programmet med en kørselsfejl.

concat(strs) strs er et vilkårligt antal strengudtryk, adskilt af kommaer. Resultatet er en streng, der består af sammensætningen af de angivne strenge. Hvis længden af resultatet er større end 255, stopper programmet med en kørselsfejl. Bemærk, at strengsammensætning også kan foretages med plus (+) operatoren. concat er udelukkende medtaget i PolyPascal for at sikre kompatibilitet med andre versioner af Pascal.

De følgende strengprocedurer er standard i PolyPascal:

delete(s,i,n) s er en strengvariabel og både i og n er udtryk af typen integer. Proceduren fjerner n tegn fra s, startende med det i'te tegn. Hvis i er større end len(s), bliver ingen tegn fjernet. Hvis i+n–1 er større end len(s), bliver len(s)–i+1 tegn fjernet. Hvis i er udenfor området 1..255, stopper programmet med en kørselsfejl.

insert(p,s,i) p er et strengudtryk, s er en strengvariabel, og i er et udtryk af typen integer. Proceduren indsætter strengen p i strengen s på den i'te position. Hvis i er større end len(s), bliver p sat i forlængelse af s. Hvis resultatet er større end den maksimale længde af s, vil s kun indeholde de første tegn. Hvis i er udenfor området 1..255, stopper programmet med en kørselsfejl.

val(s,x,p) s er et strengudtryk, x er en variabel af typen real eller af typen integer, og p er en variabel af typen integer. Den numeriske streng i s konverteres til en værdi af samme type som x, og gemmes i x. Den numeriske værdi skal følge de regler, der gælder for numeriske konstanter (se afsnit 2.2). Hverken foranstående eller efterfølgende blanktegn tillades. Hvis konverteringen forløber korrekt, sættes p til 0. Ellers sættes p til positionen på det første ulovlige tegn (i så tilfælde er værdien af x ubestemmelig).

str(p,s) p er en write–parameter af typen real eller af typen integer, og s er en strengvariabel. Proceduren konverterer den numeriske værdi til en streng og gemmer resultatet i s. Formatet af write–parametre er beskrevet i afsnit 16.3.

Det nedenfor viste program demonstrerer strengbehandling i Poly-Pascal:

```
PROGRAM strengdemo;
VAR
  st,mere,delstr,mindre: STRING[64];
  i,p: integer;
  r: real;
BEGIN
  st:='sammensat';
  mere:='dette er en '+st+' streng';
  writeln('linie 1: ',mere);
  st:='her st}r 16 tegn';
  writeln('linie 2: ',len(st),' ',len(''));
  st:='dette er en tekststreng';
  delstr:='tekst';
  writeln('linie 3: ',pos(delstr,st),' ',pos('7','12345'));
  st:='delstrenge udtages med copy funktionen';
  mindre:=copy(st,pos('u',st),7);
  writeln('linie 4: ',mindre);
  writeln('linie 5: ',copy('12345',3,255));
  st:='dette er en meget lang tekststreng';
  delete(st,pos('en',st)+3,11);
  writeln('linie 6: ',st);
  st:='a er lig med b';
  insert('mindre end eller ',st,6);
```

```
    writeln('linie 7: ',st);
    st:='-1547';
    val(st,i,p);
    writeln('linie 8: ',i,' ',p);
    r:=pi;
    str(r:10:6,st);
    writeln('linie 9: ',st);
END.
```

Programmet giver den følgende udskrift:

```
linie 1: dette er en sammensat streng
linie 2: 16 0
linie 3: 13 0
linie 4: udtages
linie 5: 345
linie 6: dette er en tekststreng
linie 7: a er mindre end eller lig med b
linie 8: -1547 0
linie 9: 3.141593
```

8.4.1 fmt funktionen

Business versionen af PolyPascal–86 indeholder en avanceret streng formateringsfunktion kaldet fmt. fmt funktionen findes ikke i standardversionen af PolyPascal, og ej heller i 8087 versionen af PolyPascal–86. Formatet af et kald af fmt er:

```
fmt(fs,v1,v2,...,vn)
```

hvor fs er et strengudtryk og v1,v2,...,vn er integer, real eller strengudtryk. Resultatet er en streng.

Formatstrengen, fs, indeholder et "billede" af den streng, der skal returneres. Længden af den returnerede streng er altid den samme som længden af fs. fs består af et antal felter, der hver bruges til at gemme et af argumenterne. Der er to typer felter: Numeriske felter og strengfelter.

Et numerisk felt er sammensat af et eller flere af de følgende tegn:

```
#  @  *  $  -  +  ,
```

Ethvert andet tegn afslutter det numeriske felt. Hvert tegn har en bestemt betydning, hvilket forklares i det følgende:

\# Angiver en cifferposition. Hvis det numeriske felt ikke indeholder '@' eller '*' tegn, bliver ubrugte cifre gemt som blanktegn. Hvis tallet er negativt og det numeriske felt ikke indeholder fortegnspositioner ('–' eller '+' tegn), bliver et flydende minustegn gemt foran tallet. Hvis tallet er for stort til at kunne udtrykkes i det angivne format, bliver alle cifferpositioner fyldt med stjerner. Eksempler:

```
fmt('####',41.578) = ' 42'
fmt('###.##',54.321) = ' 54.32'
fmt('####.##',–12.3) = ' –12.30'
fmt('###.##',1234.5) = '***.**'
```

@ Angiver en cifferposition, og indikerer, at alle ubrugte cifre skal gemmes som nuller. '@' tegnet behøver kun forekomme en gang i det numeriske felt for at frembringe denne effekt. Tallets fortegn vises ikke medmindre feltet indeholder en fortegnsposition ('–' eller '+' tegn). Eksempler:

```
fmt('@###',7) = '0007'
fmt('@@@.@@',5.678) = '005.68'
```

* Angiver en cifferposition, og indikerer, at alle ubrugte cifre skal gemmes som stjerner. '*' tegnet behøver kun forekomme en gang i det numeriske felt for at frembringe denne effekt. Tallets fortegn vises ikke medmindre feltet indeholder en fortegnsposition ('–' eller '+' tegn). Eksempler:

```
fmt('*###.##',1.492) = '***1.49'
fmt('****',100) = '*100'
```

$ Angiver en cifferposition, og indikerer, at der skal gemmes et flydende dollartegn ($) foran tallet. '$' tegnet behøver kun forekomme en gang i det numeriske felt for at frembringe denne effekt. Eksempler:

```
fmt('$#####.##',127.75) = ' $127.75'
fmt('######.#$',–10.637) = ' –$10.64'
fmt('*$####.##',49.85) = '***$49.85'
```

– Angiver en fortegnsposition. Hvis tallet er positivt, ændres minustegnet til et blanktegn; i modsat fald ændres det ikke. Eksempler:

```
fmt('–###.##',–3.5) = '– 3.50'
fmt('–###.##',10) = ' 10.00'
fmt('*#####.##–',–127.75) = '***127.75–'
```

+ Angiver en fortegnsposition. Hvis tallet er negativt, ændres plustegnet til et minustegn; i modsat fald ændres det ikke. Eksempler:

```
fmt('+###.##',–3.5) = '– 3.50'
fmt('+###.##',10) = '+ 10.00'
fmt('*$####.##+',27.75) = '***$27.75+'
```

, Angiver et decimalkomma eller et separatorkomma. Det sidste punktum eller komma i det numeriske felt antages at være decimalseparatoren.

Angiver et decimalpunktum eller et separatorpunktum. Det sidste punktum eller komma i det numeriske felt antages at være decimalseparatoren. Eksempler:

```
fmt('##,###,###.##',12345.80) = ' 12,345.80'
fmt('$#.###.###,##',–12345.80) = ' –$12.345,80'
fmt('*$,###,###.##+',12345.80) = '***$12,345.80+'
fmt('##,###.##',123456.0) = '**,***.**'
```

Et strengfelt er sammensat af et eller flere af tegnene '#' og '@'. Hvis feltet udelukkende består af '#' tegn, bliver strengen lagret venstrejusteret. Hvis der er et eller flere '@' tegn i feltet, bliver strengen lagret højrejusteret. Hvis strengen er længere end feltet, er det kun de første tegn i strengen, der lagres. Eksempler:

```
fmt('##########','Pascal') = 'Pascal '
fmt('@@@@@@@@@@','Pascal') = ' Pascal'
fmt('####','PolyPascal') = 'Poly'
fmt('@@@@','PolyPascal') = 'Poly'
```

Blanktegn og andre tegn i formatstrengen ændres ikke af fmt funktionen, og de findes derfor også i resultatet:

```
fmt('Total: $#,###.##',1234.75) = 'Total: $1,234.75'
```

Argumenterne i argumentlisten svarer til felterne i formatstrengen et for et:

```
fmt('### *###.## @###','TX',10,10) = 'TX **10.00 0010'
```

Hvis der er flere argumenter end der er felter, bliver de overskydende argumenter ignoreret. Hvis der er færre argumenter end der er felter, bliver de overskydende felter ikke ændret. Eksempler:

```
fmt('##,###.##',1000,2000) = ' 1,000.00'
fmt('#### ####',10) = ' 10 ####'
```

8.5 Strenge og tegn

Strengtyper og den standard skalare type char er kompatible, dvs. når en strengværdi forventes, kan et tegn (en char værdi) angives i stedet, og omvendt. Desuden kan strenge og tegn blandes i udtryk. Når en char variabel bliver tildelt en strengværdi, skal strengens længde være præcis 1, ellers stopper programmet med en kørselsfejl.

De enkelte tegn i en streng kan adresseres individuelt gennem indexering. Dette opnås ved at efterfølge strengvariablen med et indexudtryk, af typen integer, omsluttet af (. og .) symbolerne. Nogle eksempler:

```
linie[37]  ciffer[i]  navn[len(navn)-1]
```

Tegnet med index 0 angiver strengens nuværende længde. Således svarer len(s) til ord(s(.0.)). Der føres ikke kontrol med, om værdier, der tilskrives længdeindikatoren, er mindre end den maksimale længde af strengvariablen.

Når værdikontrol er aktiveret, dvs. når sætninger compileres i {$R+} stillingen, genereres der tillige kode, der checker, at værdier i indexudtryk ikke overstiger strengvariablenes maksimale længder. Det er imidlertid muligt at indexere en streng udover dens nuværende dynamiske længde. Indholdet af sådanne tegnpositioner er tilfældigt, og tilskrivninger til dem ændrer ikke strengens reelle værdi.

8.6 Prædefinerede strenge

PolyPascal indeholder en række prædefinerede strenge, der kan bruges til styring af diverse skærmfunktioner. Strengene defineres ved hjælp af INSTALL programmet. Nedenfor følger en beskrivelse af hvad der sker, når de udskrives:

clrhom	Clear home. Sletter skærmen og placerer cursoren i øverste venstre hjørne.
clreos	Clear to end of screen. Sletter alle tegn fra cursoren og til slutningen af skærmen.
clreol	Clear to end of line. Sletter alle tegn fra cursoren og til slutningen af den nuværende linie.
inslin	Insert line. Indsætter en blank linie på skærmen og rykker den nuværende linie, og alle under den, en linie ned.
dellin	Delete line. Sletter den nuværende linie og rykker alle linier under den en linie op. Den nederste linie på skærmen blankes.
rvson	Reverse on. De efterfølgende tegn skrives i inverteret skrift.
rvsoff	Reverse off. Ophæver rvson effekten.
inton	Intensify on. De efterfølgende tegn skrives i kraftigere (eller svagere) intensitet.
intoff	Intensify off. Ophæver inton effekten.
ulnon	Underline on. De efterfølgende tegn understreges.
ulnoff	Underline off. Ophæver ulnon effekten.
blkon	Blink on. De efterfølgende tegn blinker.
blkoff	Blink off. Ophæver blkon effekten.
alloff	All attributes off. Ophæver alle de ovenstående tegnattributer.

Bemærk, at clrhom er den eneste streng, der behøver installeres, for at PolyPascal kan anvendes. Der er altså ingen garanti for, at de andre strenge fungerer. Imidlertid vil en udefineret streng altid have længden 0, hvilket man for eksempel kan bruge i en test som vist nedenfor:

```
IF rvson='' THEN write('Inverteret skrift findes ikke');
```

9. Arraystrukturer

En arraystruktur består af et antal dataelementer, der alle er af samme type, kaldet elementtypen. Et element refereres til ved at angive et index, der bestemmer hvor i arraystrukturen, elementet findes. Indices udregnes ved hjælp af indexudtryk, og deres type kaldes indextypen.

9.1 Brug af arraystrukturer

En arraytypes definition angiver både elementtypen og indextypen (og dermed antallet af elementer i arraystrukturen). Arraytypen indledes med det reserverede ord ARRAY. Herefter følger indextypen, der er omsluttet af [og] symbolerne, og tilsidst det reserverede ord OF, efterfulgt af elementtypen. Nogle eksempler:

```
TYPE
  ciffer = 0..9;
  farve = (roed,groen,blaa);
  liste = ARRAY[1..100] OF real;
VAR
  ciffernavn: ARRAY[ciffer] OF STRING[10];
  intensitet: ARRAY[farve] OF ciffer;
  a,b: liste;
```

Et arrayelement udvælges ved at angive arrayvariablens navn efterfulgt af et indexudtryk omsluttet af [og] symbolerne. Nogle eksempler:

```
ciffernavn[5]:='fem';
intensitet[groen]:=7;
a[i]:=a[i]*2.5/b[j+1];
a:=b;
```

Da tilskrivning mellem to variable af samme type altid er tilladt, er det muligt at kopiere hele arraystrukturer ved hjælp af tilskrivningsoperatoren som vist ovenfor (a:=b).

{$R+} og {$R-} compilerdirektiverne bestemmer, om compileren skal generere kode til at foretage kørselskontrol af indexværdier. Ved start af compileren bliver {$R+} automatisk valgt, og når arrayvariable indexeres i denne stilling, bliver indexudtrykkene checket mod grænserne i deres tilhørende indextype.

I PolyPascal–80 kan man ved hjælp af {$S+} og {$S–} compilerdirektiverne bestemme, om den kode, der genereres til at udføre indexeringer af arrays, skal optimeres med hastighed eller lagerforbrug for øje. Ved start af compileren vælges {$S–}, og kode til indexering, der genereres i denne stilling, bliver optimeret med henblik på størrelsen af koden. I {$S+} stillingen bliver koden derimod optimeret med henblik på eksekveringshastighed. Hastighedsgevinsten er størst for flerdimensionale arrays. {$S+} og {$S–} har ingen effekt i PolyPascal–86, da koden her altid optimeres både med hensyn til hastighed og lagerforbrug.

9.2 Flerdimensionale arraystrukturer

En arraystrukturs elementtype kan være enhver datatype. Således kan elementtypen igen være en arraytype, og da er arraystrukturen flerdimensional. Nogle eksempler:

```
TYPE
  skakbrik = (tom,konge,dame,taarn,springer,loeber,bonde);
  skakbraet = ARRAY[1..8] OF ARRAY[1..8] OF skakbrik;
  side = 0..9;
  terning = ARRAY[side] OF ARRAY[side] OF
            ARRAY[side] OF real;
VAR
  braet: skakbraet;
  t: terning;
```

Definitionen af en flerdimensional arraystruktur kan skrives kortere ved at angive alle indextyper på en gang:

```
skakbraet = ARRAY[1..8,1..8] OF skakbrik;
terning = ARRAY[side,side,side] OF real;
```

På samme måde kan en reference til et element i en flerdimensional arraystruktur også forkortes:

```
braet[3,7]      svarer til      braet[3][7]
t[0,4,3]        svarer til      t[0][4][3]
```

En flerdimensional arraytype kan naturligvis udtrykkes ved hjælp af tidligere definerede arraytyper. Et eksempel:

```
CONST
  n = 3; m = 4;
TYPE
  vektor = ARRAY[1..n] OF real;
  matrix = ARRAY[1..m] OF vektor;
VAR
  v1,v2: vektor;
  m1: matrix;
```

I dette tilfælde er de nedenfor viste tilskrivninger tilladte:

```
v1[i]:=m1[i,j]+1.5;
m1[i][j]:=v1[i]+v2[j];
v1:=v2;
v2:=m1[i];
m1[j]:=v1;
```

9.3 Prædefinerede arraystrukturer

Til brug for direkte adgang til CPU lager og dataporte findes der i PolyPascal to prædefinerede arraystrukturer kaldet mem og port.

9.3.1 mem arraystrukturen

mem arraystrukturen giver adgang til CPU'ens lager. Til hvert element i mem svarer en byte i lageret. Syntaksen for brug af mem arraystrukturen afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
I PolyPascal–86 kræver mem arraystrukturen en segmentadresse og en offsetadresse, adskilt af et kolon, som index. Eksempler:

```
data:=mem[seg(v):ofs(v)+32];
mem[dseg:$80]:=len(s);
```

Det første udtryk angiver segmentadressen og det andet udtryk angiver offsetadressen. Begge udtryk skal være af typen integer.

PolyPascal–86 giver også mulighed for direkte lagertilgang i form af words (integers). Dette foregår ved hjælp af memw arraystrukturen. memw svarer fulstændig til mem, bortset fra, at de værdier der læses eller skrives, er words (med den mindst betydende byte først). Nogle eksempler:

```
bdos_segment:=memw[$0000:$0382];
stack_base:=memw[dseg:$15];
```

PolyPascal–80
Indextypen for mem arraystrukturen er integer. Når et element tilskrives en værdi, bliver værdien gemt i lageret på den adresse, der er givet ved indexudtrykket, og når der i et udtryk refereres til et element, bliver værdien læst fra den adresse i lageret, der svarer til indexudtrykket. Nogle eksempler:

```
mem[addr(v)+offset]:=$C0;
iobyte:=mem[3];
mem[i]:=mem[i+1];
```

9.3.2 port arraystrukturen

port arraystrukturen giver adgang til CPU'ens dataporte. Hvert element i port repræsenterer en dataport, hvis portadresse er givet ved indexudtrykket. Indextypen er byte i PolyPascal–80 og integer i PolyPascal–86, da dataportenes adresser er 8–bits for Z–80 processoren og 16–bits for 8086 processoren. Når et element i port arraystrukturen tilskrives en værdi, bliver værdien udlæst på porten (en OUT instruktion), og når der i et udtryk refereres til et element, bliver værdien læst fra porten (en IN instruktion). Nogle eksempler:

```
port[$46]:=$FF;
port[base]:=port[base] EXOR mask;
WHILE port[$B2] AND $80=0 DO {vent};
```

Visse begrænsninger knytter sig til port arraystrukturen. For det første er det ikke muligt at referere til hele arraystrukturen (reference uden indexudtryk), og for det andet kan elementer ikke anvendes som var–parametre til procedurer og funktioner.

I PolyPascal–86 findes også en portw arraystruktur, der anvendes til 16–bits port I/O. portw svarer fuldstændig til port, bortset fra, at der ind– og udlæses 16–bits værdier.

9.4 Tegn–arrays

Tegn–arrays er arraystrukturer med et index og elementer af typen char, dvs. arraystrukturer der er defineret som:

```
ARRAY[n..m] OF char
```

hvor n og m er af typen integer, og m er større end n. Tegn–arrays kan opfattes som strenge med en konstant længde (m–n+1). Strengkonstanter kan tilskrives til tegn–arrays, forudsat at længden af konstanten er lig med antallet af elementer i arraystrukturen (m–n+1).

I PolyPascal kan tegn–arrays indgå i strengudtryk. Et tegn–array konverteres da til en streng af længden m–n+1 (hvor n og m er nedre og øvre grænse i tegn–arrayets indextype). Tegn–arrays kan altså sammenlignes og manipuleres på samme måde som strenge. Bemærk dog, at værdier udregnet i strengudtryk ikke kan tilskrives til tegn–arrays – de eneste værdier, der kan tilskrives, er strengkonstanter og andre tegn–array variable af samme type.

10. Poster

En post er en struktureret datatype, og på samme måde som en array-struktur består den af flere enkelte elementer. I en post kaldes elementerne felter. I et felt kan der lagres værdier af en bestemt datatype, der angives i definitionen af posttypen. Datatypen kan vælges helt frit, og kan eventuelt være endnu en posttype.

10.1 Brug af poster

En posttypes definition angiver en datatype og et feltnavn for hvert felt i posten. Definitionen indledes med det reserverede ord RECORD og afsluttes med det reserverede ord END. Herimellem skrives en feltliste, der består af en eller flere feltsektioner, adskilt af semikoloner. Hver feltsektion består af en eller flere identifiere, adskilt af kommaer, efterfulgt af et kolon og en datatype. Nogle eksempler:

```
TYPE
  complex = RECORD
              re,im: real;
            END;
  maanedsnavn = (jan,feb,mar,apr,maj,jun
                 jul,aug,sep,okt,nov,dec);
  datopost = RECORD
               dag: 1..31;
               maaned: maanedsnavn;
               aar: 1900..1999;
             END;
  navnepost = RECORD
                efternavn: STRING[32];
                antalnavne: 1..3;
                fornavne: ARRAY[1..3] OF STRING[16];
              END;
VAR
  x,y: complex;
  dato: datopost;
  kunde: navnepost;
  fridage: ARRAY[1..14] OF datopost;
```

Identifierne re, im, dag, maaned, aar, efternavn, antalnavne og fornavne er feltnavne. Et feltnavn er lokalt for en posttype, og samme feltnavn kan derfor anvendes i flere forskellige posttyper. Et felt i en post angives ved at skrive et punktum og feltnavnet efter postvariablen. Nogle eksempler:

```
x.re:=5.9;
y.im:=x.im+y.im;
dato.dag:=2;
dato.maaned:=dec;
dato.aar:=1960;
kunde.efternavn:='Hansen';
kunde.antalnavne:=2;
kunde.fornavne[1]:='Karl';
kunde.fornavne[2]:='Erik';
fridage[10]:=dato;
x:=y;
```

Bemærk, at samtlige værdier i en post kan kopieres direkte over i de tilsvarende felter i en anden post af samme type, som vist i de sidste to tilskrivninger ovenfor.

Felter i en post kan være af enhver datatype. Således kan en felttype være yderligere en posttype, som vist nedenfor:

```
TYPE
  personpost = RECORD
                 navn: navnepost;
                 foedselsdato: datopost;
               END;
VAR
  p1,p2,p3: personpost;
```

I dette tilfælde er de nedenfor viste tilskrivninger tilladte:

```
p1.navn.efternavn:='Jensen';
p1.navn.antalnavne:=1;
p1.navn.fornavne[1]:='Peter';
p1.foedselsdato.dag:=17;
p1.foedselsdato.maaned:=mar;
p1.foedselsdato.aar:=1951;
p2.navn:=p1.navn;
p3:=p2;
```

10.2 WITH sætninger

Den ovenfor anvendte notation kan være en smule anstrengende, men den kan da også forkortes ved hjælp af en WITH sætning. WITH sætningen "åbner" en post, således at feltnavnene bliver tilgængelige som almindelige variabelidentifiere. En WITH sætning starter med det reserverede ord WITH. Herpå følger en række postvariable adskilt af kommaer, efterfulgt af det reserverede ord DO og en sætning. I den indskrevne sætning kan feltidentifierne i de postvariable, der angives mellem WITH og DO, anvendes som almindelige variabelidentifiere. Den nedenfor viste WITH sætning svarer til sætningerne ovenfor:

```
WITH p1,navn,foedselsdato DO
BEGIN
  efternavn:='Jensen';
  antalnavne:=1;
  fornavne[1]:='Peter';
  dag:=17;
  maaned:=mar;
  aar:=1951;
  p2.navn:=navn;
  p3:=p2;
END;
```

Antag at den følgende erklæring er foretaget:

```
VAR
  datoer: ARRAY[1..7] OF datopost;
```

Sætningen:

```
WITH datoer[4] DO
IF maaned=dec THEN
BEGIN
  maaned:=jan; aar:=aar+1;
END ELSE maaned:=succ(maaned);
```

svarer da til:

```
IF datoer[4].maaned=dec THEN
BEGIN
  datoer[4].maaned:=jan;
  datoer[4].aar:=datoer[4].aar+1;
END ELSE
datoer[4].maaned:=succ(datoer[4].maaned);
```

Formen:

```
WITH p1,p2,....,pn DO
```

svarer til:

```
WITH p1 DO WITH p2 DO .... WITH pn DO
```

I PolyPascal–80 bestemmes det maksimale antal nestede WITH sætninger, der tillades, dvs. det maksimale antal poster, der kan åbnes på en gang, via compilerens W register. Ved start af compileren vælges {$W4}, hvilket betyder, at maksimalt 4 WITH sætninger kan nestes. Det maksimale nestingniveau kan varieres mellem 0 og 9, og er givet ved indholdet af W registeret ved begyndelsen af den nuværende blok. Hvis et {$W8} compilerdirektiv placeres før erklæringsdelen i en blok, kan WITH sætninger i denne blok altså nestes op til 8 gange (bemærk, at dette ikke betyder, at der kun må være 8 WITH sætninger i blokken, men derimod, at der højst må være 8 WITH sætninger inden i hinanden). I begyndelsen af en blok reserverer compileren 2 bytes for hver tilladt nestingniveau.

W compilerregisteret har ingen betydning i PolyPascal–86, men W compilerdirektiver tillades alligevel. Det maksimale nestingniveau i PolyPascal–86 er altid 16.

10.3 Variant–del i poster

En posttype kan defineres sådan, at forskellige variable af denne posttype kan indeholde forskellige felter. Dette opnås ved at indføje en variant–del i posttypen. Variant–delen indledes med det reserverede ord CASE, efterfulgt af en skalar datatype, kaldt mærketypen, igen efterfulgt af det reserverede ord OF. Hver variant startes med en eller flere konstanter, hvis type er givet ved mærketypen, efterfulgt af et kolon. Selve varianten angives som en feltliste omsluttet af parenteser. Varianterne adskilles med semikoloner.

Antag, som et eksempel, at man ønsker at beskrive tre forskellige geometriske figurer ved hjælp af en enkelt datatype. Hvis de tre geometriske figurer er:

```
TYPE
  figurtype = (rektangel,triangel,cirkel);
```

Så kan den det generelle format af den beskrivende type for eksempel være:

```
TYPE
  figur = RECORD
            <felter fælles for alle figurer>;
            CASE figurtype OF
              rektangel: (<felter for rektangler>);
              triangel: (<felter for triangler>);
              cirkel: (<felter for cirkler>);
          END;
```

Almindeligvis indeholder en post med en variant–del et felt, der angiver hvilken variant, der bruges i øjeblikket. Dette felt kaldes mærkefeltet. Mærkefeltet for typen figur kunne for eksempel være:

```
art: figurtype
```

Når en posttype defineres, skrives mærkefeltet imellem de reserverede ord CASE og OF:

```
TYPE
  figur = RECORD
            farve: (roed,groen,blaa);
            CASE art: figurtype OF
              rektangel: (hoejde,bredde: real);
              triangel: (side1,side2,vinkel: real);
              cirkel: (radius: real);
          END;
```

Den faste del af en posttype skal altid skrives før variant–delen. I typen figur er feltet farve det eneste felt i den faste del. En posttype kan kun indeholde en variant–del. En tom variant–del skrives som to parenteser, der ikke omslutter noget.

Bemærk at PolyPascal systemet ikke fører kontrol med, om man i en variant–del, der er beregnet for en type af information, lagrer information af en anden type. I en variabel af den overfor viste type figur giver det derfor ikke fejl at referere til feltet radius, selvom mærkefeltet art ikke er lig med værdien cirkel. Faktisk kan mærkefeltet helt udelades, således at der kun angives en datatype mellem CASE og OF.

Længden, af en post med variantdel, er givet ved længden af den faste del plus længden af mærkefeltet (hvis det eksisterer) plus længden af den største variant.

11. Mængder

En samling af objekter, der refereres til under et, kaldes en mængde. Objekterne, der indgår i mængden, kaldes mængdens elementer. Nogle eksempler på mængder:

A. Alle heltal mellem 0 og 100.
B. Primtallene mellem 0 og 100.
C. Alfabetets bogstaver.
D. Alfabetets vokaler.

To mængder er ens hvis og kun hvis de indeholder de samme elementer. Rækkefølgen af elementerne har ingen betydning, så mængderne [1,3,5] og [5,3,1] er altså ens. Hvis elementerne i en mængde også indgår i en anden mængde, er den første mængde en delmængde af den anden. For de ovenfor nævnte mængder gælder der, at B er en delmængde af A og D er en delmængde af C. D er også en delmængde af sig selv, men ikke en ægte delmængde, idet der samtidigt gælder, at D er lig med D.

I Pascal findes der tre forskellige mængdeoperationer. Foreningsmængden af to mængder A og B, hvilket skrives som A+B, er den mængde, der indeholder alle elementer fra A og alle elementer fra B. For eksempel er foreningsmængden af [1,3,5,7] og [2,3,4] lig med [1,2,3,4,5,7]. Fællesmængden af to mængder A og B, hvilket skrives som A*B, er den mængde, hvis elementer både er elementer i A og B. For eksempel er fællesmængden af [1,3,5,7] og [2,3,4] lig med [3]. Differensmængden af to mængder A og B, hvilket skrives som A–B, er den mængde, hvis elementer er elementer i A men ikke i B. For eksempel er differensmængden af [1,3,5,7] og [2,3,4] lig med [1,5,7].

11.1 Mængdetyper

Teoretisk set er der ingen grænse for, hvormange elementer en mængde kan indeholde, ligesom elementer i en mængde i teorien kan være hvad som helst. En mængde i Pascal kan derimod kun indeholde et vist antal elementer, og elementerne skal alle være af samme type (kaldet elementtypen). Elementtypen må være enhver skalar type, undtagen real. En mængdetype angives ved de reserverede ord SET OF efterfulgt af elementtypen. Nogle eksempler:

```
TYPE
  talmaengde = SET OF 0..50;
  cifre = SET OF 0..9;
  bogstaver = SET OF 'A'..'Z';
  farver = SET OF (roed,groen,blaa);
  tegnsaet = SET OF char;
```

Det maksimale antal elementer, en mængde i PolyPascal kan indeholde, er 256. Desuden gælder der, at den ordinale værdi af grænserne i elementtypen skal være mellem 0 og 255.

11.2 Mængdeudtryk

Mængder kan udregnes udfra andre mængder ved hjælp af mængdeudtryk. Disse opbygges af mængdekonstanter, mængdevariable, mængdeangivere og operatorer.

11.2.1 Mængdeangivere

En mængdeangiver består af en liste af elementer og intervaller omsluttet af [og] symbolerne. En element er et udtryk af mængdens elementtype, og et interval er to sådanne udtryk adskilt af et dovent kolon (..). Nogle eksempler på mængdeangivere:

```
[1,3,5]
['P','O','L','Y']
[x]
[i,j,k+3]
[1..5]
[i..j]
['A'..'Z','a'..'z','0'..'9']
[1,3..10,12]
```

Mængden [1..5] svarer til [1,2,3,4,5]. Hvis i er større end j så angiver [i..j] en tom mængde. [] angiver også en tom mængde, men denne tomme mængde er kompatibel med alle mængdetyper, da den ikke indeholder nogen udtryk, der kan bestemme dens type.

11.2.2 Mængdeoperatorer

I mængdeudtryk findes der tre prioritetsniveauer for operatorer. Fællesmængdeoperatoren (*) har den højeste prioritet, efterfulgt af forenings- og differensmængdeoperatorerne (+ og -), igen efterfulgt af de relationelle operatorer og IN operatoren. Her følger en oversigt over mængdeoperatorerne:

*	Fællesmængde.
+	Foreningsmængde.
-	Differensmængde.
=	Test for lighed.
<>	Test for ulighed.
>=	Sandt hvis den anden operand er en delmængde af den første.
<=	Sandt hvis den første operand er en delmængde af den anden.
IN	Den anden operand er en mængde, og den første operand er et udtryk af samme type som mængdens elementtype. Resultatet er sand hvis elementet, der er givet ved det første udtryk, er indeholdt i mængden.

Mængdeudtryk kan ofte anvendes til at tydeliggøre lange og indviklede sandhedsudtryk. For eksempel svarer konstruktionen:

```
IF (ch='E') OR (ch='C') OR (ch='S') OR (CH='I') THEN
```

til den noget mere elegante konstruktion:

```
IF ch IN ['E','C','S','I'] THEN
```

På tilsvarende måde kan konstruktionen:

```
IF (ch>='0') AND (ch<='9') THEN
```

forkortes til:

```
IF ch IN ['0'..'9'] THEN
```

11.3 Mængdetilskrivninger

Tilskrivningsoperatoren kan anvendes til at tilskrive mængdevariable mængdeværdier, der er udregnet i mængdeudtryk. Eksempler:

```
VAR
  cifre: SET OF 0..9;
  vokaler,konsonanter: SET OF 'A'..'Z';
  ulige: SET OF 1..100;
  i: integer;
BEGIN
  cifre:=[0..9];
  vokaler:=['A','E','I','O','U','Y'];
  konsonanter:=['A'..'Z']-vokaler;
  ulige:=[];
  FOR i:=1 TO 50 DO ulige:=ulige+[i+i-1];
END.
```

12. Typeangivne konstanter

I modsætning til definitionen af en typeløs konstant (se afsnit 4.2.2), specificerer definitionen af en typeangiven konstant både konstantens type og dens værdi. En typeangiven konstant svarer til en variabel af samme type, blot med en konstant værdi.

Typeangivne konstanter defineres i en konstantdefinitionsdel i en bloks erklæringsdel. Konstantens identifier skal efterfølges af et kolon og en type, der igen skal efterfølges af et lighedstegn og selve konstanten.

12.1 Typeangivne konstanter af simple typer

En typeangiven konstant kan anvendes nårsomhelst en variabel af den samme type forventes. Typeangivne konstanter kan altså, i modsætning til almindelige typeløse konstanter, bruges som var–parametre til procedurer og funktioner. Nogle eksempler på typeangivne konstanter af simple typer:

```
CONST
  maximum: integer = 9999;
  faktor: real = -6.32774526;
  navn: STRING[10] = 'PolyPascal';
  crlf: STRING[2] = ^M^J;
```

En typeangiven konstant kan anbefales frem for en typeløs konstant, hvis konstanten anvendes ofte i programmet. Grunden hertil er, at en typeangiven konstant kun inkluderes en enkelt gang i programmets kode, medens en typeløs konstant inkluderes hver gang, der refereres til den. Da en typeangiven konstant faktisk er en variabel med en konstant værdi, kan den ikke anvendes i definitionen af andre konstanter eller typer. Det nedenfor viste eksempel er således ikke tilladt (da min og max ikke er typeløse konstanter):

```
CONST
  min: integer = 0;
  max: integer = 100;
TYPE
  liste = ARRAY[min..max] OF real;
```

12.2 Strukturerede konstanter

Ved hjælp af typeangivne konstanter er det muligt at definere konstanter af strukturerede typer, dvs. array–, post– og mængdekonstanter. Sådanne konstanter bruges normalt som prædefinerede tabeller og mængder i konverteringer og tests.

12.2.1 Arraykonstanter

En arraykonstant består af en liste af konstanter adskilt af kommaer og omsluttet af parenteser. Nogle eksempler:

```
TYPE
  farve = (roed,groen,blaa);
  navneliste = ARRAY[farve] OF STRING[4];
CONST
  farvenavn: navneliste = ('rød','grøn','blå');
  fak: ARRAY[1..7] OF integer = (1,2,6,24,120,720,5040);
```

Den ovenfor viste konstantdefinitionsdel definerer en arraykonstant, kaldet farvenavn, der bruges til at omsætte værdier af typen farve til deres tilsvarende tekstrepræsentation, og en arraykonstant, kaldet fak, der bruges til hurtige beregninger af fakulteten af et heltal. Elementerne i farvenavn er:

```
farvenavn[roed] = 'rød'
farvenavn[groen] = 'grøn'
farvenavn[blaa] = 'blå'
```

Flerdimensionale arraystrukturer tillades naturligvis også. Sådanne konstanter defineres ved at omslutte hver dimension med parenteser. Den inderste konstant svarer til dimensionen længst mod højre i arraystrukturen. Nogle eksempler:

```
TYPE
  matrix = ARRAY[1..2,1..3] OF real;
  kubus = ARRAY[0..1,0..1,0..1] OF integer;
CONST
  m: matrix = ((1.0,1.1,1.2),(6.0,7.5,9.0));
  k: kubus = (((1,7),(2,9)),((0,8),(6,3)));
```

Elementerne i k er:

```
k[0,0,0] = 1      k[0,0,1] = 7
k[0,1,0] = 2      k[0,1,1] = 9
k[1,0,0] = 0      k[1,0,1] = 8
k[1,1,0] = 6      k[1,1,1] = 3
```

Elementtypen i en arraykonstant kan være enhver type undtagen filtyper og pointertyper. Tegn–array konstanter angives enten som enkelttegn eller som strenge. Definitionen:

```
CONST
  cifre: ARRAY[0..9] OF char =
    ('0','1','2','3','4','5','6','7','8','9');
```

kan altså også skrives som:

```
CONST
  cifre: ARRAY[0..9] OF char = '0123456789';
```

12.2.2 Postkonstanter

En postkonstant består af en liste af feltkonstanter adskilt af semikoloner og omsluttet af parenteser. Hver feltkonstant angiver en feltidentifier efterfulgt af et kolon og en konstant. Nogle eksempler:

```
TYPE
  complex = RECORD
              re,im: real;
              END;
  maanedsnavn = (jan,feb,mar,apr,maj,jun,
                 jul,aug,sep,okt,nov,dec);
  datopost = RECORD
               dag: 1..31;
               maaned: maanedsnavn;
               aar: 1900..1999;
             END;
  navnepost = RECORD
                efternavn: STRING[32];
                antalnavne: 1..3;
                fornavne: ARRAY[1..3] OF STRING[16];
              END;
```

```
CONST
  c1: complex = (re: 4.75; im: -8.0);
  kdato: datopost = (dag: 12; maaned: feb; aar: 1983);
  hansen: navnepost = (efternavn: 'Hansen';
                       antalnavne: 2;
                       fornavne: ('Jens','Erik',''));
  tal: ARRAY[1..3] OF complex =
    ((re: 1; im: 2),(re: 0; im: 37),(re: -3; im: 8.5));
```

Feltkonstanter skal angives i den rækkefølge, felterne er erklæret i postens definition. Hvis en post indeholder felter af filtyper eller pointertyper, er det ikke muligt at definere konstanter af denne posttype. Hvis en post indeholder en variant, er det op til programmøren kun at angive felter, der indgår i den valgte variant. Hvis varianten indeholder et mærkefelt, skal en værdi også angives for dette.

12.2.3 Mængdekonstanter

Syntaksen for en mængdekonstant er nøjagtig den samme som for en mængdeangiver (se afsnit 11.2.1), bortset fra, at elementerne og intervallernes grænser naturligvis skal være konstanter. Nogle eksempler på mængdekonstanter:

```
TYPE
  cifre = SET OF 0..9;
  bogstaver = SET OF 'A'..'Z';
CONST
  lige: cifre = [0,2,4,6,8];
  vokaler: bogstaver = ['A','E','I','O','U','Y'];
  alfanum: SET OF char = ['A'..'Z','a'..'z','0'..'9'];
```

13. Filer

Ved hjælp af filer kan et program gemme data, der senere kan læses af programmet selv eller af et andet program. I modsætning til andre datatyper, bliver data, der udlæses til en fil, ikke gemt i lageret, men udskrives i stedet til en diskfil. På samme måde bliver data, der indlæses fra en fil, læst fra en diskfil.

En fil er en følge af elementer, alle af samme datatype. Antallet af elementer i en fil, ofte kaldet længden af filen, er ikke på forhånd fastsat i typedefinitionen, som det for eksempel er tilfældet med en arraystruktur.

Til hver fil hører der en fil–pointer, der kontrolleres af systemet. Hver gang et element udlæses til eller indlæses fra en fil, flytter systemet automatisk fil–pointeren frem til det næste element i filen. Systemet er desuden i stand til at flytte fil–pointeren til ethvert af brugeren valgt element i filen.

13.1 Filtyper

En filtype angives med de reserverede ord FILE OF efterfulgt af typen af filens elementer. Nogle eksempler:

```
TYPE
  keystr = STRING[8];
  data = RECORD
           key: keystr;
           navn: STRING[40];
           pris: real;
         END;
VAR
  keyfil: FILE OF keystr;
  datafil: FILE OF data;
```

Elementtypen kan være enhver datatype, undtagen en filtype (FILE OF FILE typer tillades altså ikke). Filvariable må ikke optræde i tilskrivningssætninger eller i udtryk.

I MS–DOS versionen af PolyPascal–86 definerer F compilerdirektivet, hvor mange filer der kan være åbne på samme tid. Compileren vælger selv {$F16}, hvilket betyder, at maksimalt 16 filer kan være åbne på en gang. Hvis man, for eksempel, placerer et {$F32} compilerdirektiv i begyndelsen af et program (bemærk: før erklæringsdelen), kan der være op til 32 filer åbne samtidig. F direktivet begrænser ikke antallet af filer der kan erklæres i et program – det sætter kun en grænse for hvor mange filer der kan være åbne samtidig.

Bemærk, at selvom man har anvendt et F direktiv til at sikre et tilstrækkeligt antal filbuffere, kan man stadig opleve, at systemet rapporterer en 'For mange åbne filer' fejl. I så tilfælde er det fordi MS–DOS operativsystemet er løbet tør for filbuffere. Dette kan ændres ved at angive en større værdi for 'files=xx' parameteren i CONFIG.SYS filen. MS–DOS vælger normalt 8 hvis man ikke angiver nogen værdi. Yderligere beskrivelse heraf findes i MS–DOS operativsystemets dokumentation.

13.2 Operationer på filer

Betegnelsen "filnavn" anvendes i det følgende om et MS–DOS eller CP/M disk fil navn. Formatet af et filnavn afhænger af, hvilken version af PolyPascal der anvendes.

MS–DOS versionen
I MS–DOS versionen er et filnavn faktisk en række af biblioteksnavne, der fører frem til det ønskede underbibliotek, efterfulgt af selve filnavnet:

<disk>:\<dirnavn>\...\<dirnavn>\<filnavn>

Hvis rækken starter med en baglæns skråstreg (eller på nogle maskiner et stort Ø), starter MS–DOS søgningen i diskens hovedbibliotek; i modsat fald starter søgningen i det nuværende underbibliotek.

<disk> er diskens identifier (A–O). Hvis <disk> og det efterfølgende kolon udelades, vælges den nuværende disk. <dirnavn> og <filnavn> er disk fil navne i formatet:

<navn>.<type>

hvor <navn> er op til otte bogstaver eller talcifre og <type> er op til tre bogstaver eller talcifre. Hvis punktummet og <type> feltet udelades, vælger systemet automatisk en filetype der består af tre blanktegn.

CP/M versionerne
Det generelle format af et CP/M filnavn er:

<disk>:<navn>.<type>

hvor <disk> er diskens identifier (A–P), <navn> er filnavnet (op til otte bogstaver eller talcifre), og <type> er filtypen (op til tre bogstaver eller talcifre). Hvis <disk> og det efterfølgende kolon udelades, vælges den nuværende disk. Hvis punktummet og <type> feltet udelades, vælger systemet automatisk en filtype der består af tre blanktegn.

Følgende standardprocedurer kan bruges til behandling af filer (f angiver en fil–identifier):

assign(f,s) s er et strengudtryk, der angiver et MS–DOS eller CP/M filnavn. Den maksimale længde af filnavnet er 63 tegn i MS–DOS versionen. Filnavnet tilskrives til filvariablen, og alle efterfølgende filoperationer på f vil arbejde på diskfilen ved navn s. assign må aldrig anvendes på en fil, der er åben.

rewrite(f) Denne procedure bruges til at oprette nye diskfiler. Diskfilen får det navn, der er tilskrevet f, og fil–pointeren sættes til begyndelsen af filen. Den nye fil indeholder ingen elementer. Hvis der i forvejen findes en fil af samme navn, bliver denne slettet.

reset(f) Denne procedure bruges til at åbne en eksisterende diskfil. Diskfilen er givet ved det navn, der er tilskrevet f. Hvis diskfilen ikke eksisterer, rapporteres en I/O fejl – ellers åbnes filen, og fil– pointeren sættes til det første element.

read(f,vs) vs er en eller flere variable, af samme type som f's elementer, adskilt af kommaer. Hver variabel læses fra filen, og efter hver læsning flyttes fil–pointeren frem til det næste element.

write(f,vs) vs er en eller flere variable, af samme type som f's elementer, adskilt af kommaer. Hver variabel skrives til filen, og efter hver skrivning flyttes fil–pointeren frem til det næste element.

seek(f,n) n er et udtryk af typen integer. Fil–pointeren flyttes til det n'te element i filen. Det første element i en fil har nummer 0. Bemærk, at det ikke er muligt at flytte ud over længden af en fil. I MS–DOS versionen af PolyPascal–86 kan n også være et udtryk af typen real, hvilket giver et større virkeområde.

truncate(f) Kun MS–DOS versionen. Et kald til denne procedure sætter end–of–file til fil–pointerens nuværende position, dvs. "bortskærer" resten af filen.

flush(f) Et kald til flush tømmer f's interne buffer, dvs. sikrer, at f's interne buffer er udskrevet på disken. Envidere sikres det, at den næste read– operation vitterligt foretager en læsning på disken. Normalt anvendes denne procedure kun i programmer, der skal køres på flerbrugersystemer, hvor flere brugere har adgang til den samme fil på en gang. Et kald til flush har ingen effekt på typeangivne filer i MS–DOS versionen af PolyPascal–86, da der ikke findes nogen buffer i denne versions filvariable.

close(f) Diskfilen, der behandles via f, lukkes, og diskens bibliotek opdateres, så det nu angiver filens nye status.

erase(f) Diskfilen, der er sammenknyttet med f, slettes. Hvis filen, der skal slettes, er åben, dvs. hvis der er foretaget kald til rewrite eller reset men ikke til close, bør close kaldes først.

rename(f,s) Denne procedure ændrer navnet på den diskfil, der er sammenknyttet med f, til det navn, der er givet ved strengudtrykket s. rename bør ikke anvendes på åbne filer. Hvis f anvendes efter rename, uden forudgående kald til assign, arbejdes der på filen med det nye navn.

chdir(s) Kun MS–DOS versionen. s er et strengudtryk, der angiver et underbiblioteks filnavn. Dette underbibliotek bliver det nuværende underbibliotek. Hvis filnavnet angiver en disk, bliver denne disk den nuværende disk.

mkdir(s) Kun MS–DOS versionen. s er et strengudtryk, der angiver et underbiblioteks filnavn. Det sidste element i filnavnet skal angive et ikke eksisterende bibliotek, og et bibliotek med dette navn oprettes.

rmdir(s) Kun MS–DOS versionen. s er et strengudtryk, der angiver et underbiblioteks filnavn. Det sidste bibliotek i filnavnet slettes.

gcdir(d,s) Kun MS–DOS versionen. d er et heltalsudtryk og s er en strengvariabel. s returnerer det nuværende bibliotek for det drive, der udvælges af d. 0 svarer til drive A, 1 til drive B, osv.

Følgende standardfunktioner kan bruges til behandling af filer:

eof(f) Returnerer værdien sand (true) hvis f's fil–pointer er placeret ved slutningen af filen, altså "bagved" det sidste element i filen. Hvis fil– pointeren ikke er ved slutningen af filen, returneres værdien falsk (false).

position(f) En heltalsfunktion, der returnerer fil–pointerens værdi, altså nummeret på det nuværende element i filen. 0 svarer til det første element.

length(f) En heltalsfunktion, der returnerer filens længde, altså antallet af elementer i filen. Hvis length(f) er nul, er filen tom.

longpos(f) Kun MS–DOS versionen. Denne funktion svarer til position funktionen, bortset fra, at typen af den returnerede værdi er real.

longlen(f) Kun MS–DOS versionen. Denne funktion svarer til length funktionen, bortset fra, at typen af den returnerede værdi er real.

Der skal altid udføres et kald til assign, før der foretages operationer på en fil. Hvis der skal foretages læsninger og/eller skrivninger fra/til filen, skal der desuden udføres et kald til rewrite eller reset. Efter et sådant kald peger fil–pointeren på det første element i filen, altså position(f)=0. Efter et kald til rewrite gælder der desuden, at length(f)=0.

En fil kan kun udvides ved at føje elementer til slutningen af den. Fil–pointeren kan flyttes til slutningen af filen ved at udføre sætningen:

```
seek(f,length(f));
```

Hvis der er foretaget læsninger og/eller skrivninger på en fil, skal close kaldes efter endt behandling af filen. Undlades dette, kan data gå tabt, da systemet ikke får opdateret filens status korrekt vis.

Det nedenfor viste program udlæser 1000 tilfældige heltal, mellem 0 og 999, til diskfilen A:RANDOM.DAT.

```
PROGRAM skrivtal;
VAR
  i,k: integer;
  talfil: FILE OF integer;
BEGIN
  assign(talfil,'A:RANDOMS.DAT');
  rewrite(talfil);
  FOR i:=1 TO 1000 DO
  BEGIN
    k:=random(1000); write(talfil,k);
  END;
  close(talfil);
END.
```

Det nedenfor viste program indlæser de tilfældige tal fra diskfilen, der blev oprettet af det ovenstående program, og beregner deres gennemsnit:

```
PROGRAM laestal;
VAR
  i,n: integer;
  sum: real;
  talfil: FILE OF integer;
BEGIN
  assign(talfil,'A:RANDOMS.DAT');
  reset(talfil);
  sum:=0.0; n:=0;
  WHILE NOT eof(talfil) DO
```

```
    BEGIN
      read(talfil,i); sum:=sum+i; n:=n+1;
    END;
    close(talfil);
    writeln('gennemsnittet er ',sum/n:0:3);
  END.
```

Det nedenfor viste program opretter en ny fil A:TEMP.DAT, og udlæser alle elementer fra A:RANDOM.DAT til den, i omvendt rækkefølge. Derefter slettes A:RANDOMS.DAT, og A:TEMP.DAT omdøbes i stedet til dette navn. Bemærk brugen af seek proceduren og length funktionen til at læse inputfilen baglæns:

```
PROGRAM omvend;
VAR
  p,n: integer;
  indfil,udfil: FILE OF integer;
BEGIN
  assign(indfil,'A:RANDOMS.DAT'); reset(indfil);
  assign(udfil,'A:TEMP.DAT'); rewrite(udfil);
  FOR p:=length(indfil)-1 DOWNTO 0 DO
  BEGIN
    seek(indfil,p); read(indfil,n); write(udfil,n);
  END;
  close(indfil); close(udfil);
  erase(indfil); rename(udfil,'A:RANDOMS.DAT');
END.
```

I MS-DOS udregnes antallet af elementer i en diskfil fra filens længde (i bytes), der er lagret i diskens bibliotek. Hvis det sidste element i filen ikke er fyldt helt ud, bliver det ikke medtaget af length funktionen. length funktionen returnerer kun antallet af helt fyldte elementer. Hvis man forsøger at læse et ukomplet element, giver systemet en I/O fejl. Dette problem opstår imidlertid aldrig hvis en given diskfil altid behandles via filvariable af samme type.

13.3 Textfiler

Elementerne i en textfil er ASCII tegn, og hvert tegn optager en byte i filen. Forskellen på en textfil og en selvdefineret filtype er, at textfilen, udover at være en følge af elementer (tegn), yderligere er inddelt i linier. Hver linie består af en vilkårligt antal tegn afsluttet af et end-of-line mærke. MS-DOS og CP/M operativsystemerne anvender en

CR/LF sekvens (et CR tegn efterfulgt af et LF tegn) som end–of–line mærke og et Ctrl–Z tegn som end–of–file mærke. Da længden af en linie kan variere, kan positionen på en given linie ikke beregnes, og det er derfor ikke muligt at foretage tilfældig flytning af en textfils fil–pointer. Desuden er det ikke tilladt at udlæse til og indlæse fra en textfil på en gang.

En textfilvariabel erklæres ved at referere til det reserverede ord TEXT, for eksempel:

```
VAR
  indfil,udfil: TEXT;
```

En textfilvariabel indeholder bl.a. en buffer. Denne buffer er normalt på 128 bytes, hvilket betyder, at for hver 128 tegn der læses fra eller skrives til filen, foretages der en læsning fra eller skrivning til diskfilen. I de fleste tilfælde er 128 bytes en passende bufferstørrelse, men hvis et program skal kopiere data fra en textfil til en anden, kan en større buffer være at ønske; ellers vil disken bruge en masse tid på at flytte læse/skrive hovedet frem og tilbage blot for at læse eller skrive en lille blok. PolyPascal–86 tillader derfor, at man angiver bufferens størrelse når man erklærer en textfil. Dette foregår ved at skrive bufferstørrelsen i firkantparenteser umiddelbart efter TEXT:

```
VAR
  outfile: TEXT[4096];
```

Den ovenfor viste erklæring angiver at variablen outfile skal have en bufferstørrelse på 4096 (4K) bytes.

Bemærk, at PolyPascal–80 ikke giver mulighed for at ændre en textfils bufferstørrelse. Bufferstørrelsen er altid 128 bytes i PolyPascal–80.

13.3.1 Operationer på textfiler

I lighed med selvdefinerede filer skal læsninger og skrivninger til og fra en textfil altid forudgås af et kald til assign efterfulgt af et kald til reset eller rewrite. Hvis en ny textfil skal oprettes, skal rewrite bruges til at klargøre filen. I dette tilfælde tillader systemet kun udlæsning til filen. Hvis en eksisterende textfil skal læses, skal reset bruges til at klargøre filen. I dette tilfælde tillader systemet kun indlæsning fra filen. Når en textfil, der blev klargjort med rewrite, lukkes, ved et kald til close proceduren, føjer systemet automatisk et end–of–file mærke til filen.

MS–DOS versionen af PolyPascal–86 har en ekstra fil klargørings procedure ved navn append. Formatet af en kald af append er:

append(t)

hvor t er en textfilvariabel. Diskfilen med det navn, der er tilskrevet t, åbnes, og fil–pointeren flyttes til slutningen af filen. I lighed med rewrite tillader systemet kun udlæsning til filen efter et kald til append.

Tegn skrives til en textfil med write proceduren og læses fra en textfil med read proceduren. Pascal har desuden en række specielle procedurer og functioner, der muliggør behandling af linier i en textfil (t angiver en textfil–identifier):

readln(t) Flytter fil–pointeren frem til begyndelsen af den næste linie ved at læse alle tegn til og med den næste CR/LF sekvens.

writeln(t) Skriver et end–of–line mærke (en CR/LF sekvens) til filen.

eof(t) En boolean funktion der returnerer sand hvis fil–pointeren er placeret ved slutningen af filen, dvs. enten ved den fysiske slutning af filen eller ved et Ctrl–Z tegn.

eoln(t) En boolean funktion der returnerer sand hvis fil–pointeren er placeret ved slutningen af en linie (nærmere bestemt ved CR tegnet i den CR/LF sekvens der afslutter linien). eoln(t) returnerer desuden sand hvis eof(t) er sand.

seof(t) Skip og test for end–of–file. Denne funktion svarer til eof funktionen, bortset fra, at den læser og ignorerer blanktegn, TAB tegn, og end–of–line mærker (CR/LF sekvenser), før den tester for end–of–file.

seoln(t) Skip og test for end–of–line. Denne funktion svarer til eoln funktionen, bortset fra, at den læser og ignorerer blanktegn og TAB tegn før den tester for end–of–line.

De ovennævnte operationer kan kun anvendes på textfiler. seek proceduren og position og length funktionerne kan ikke anvendes på textfiler. Et kald til flush proceduren har ingen effekt på textfiler i CP/M

versionerne. I MS-DOS versionen vil et sådant kald tømme filbufferen, hvis filen anvendes til udlæsning. Når truncate proceduren anvendes på en textfil, bliver textfilen samtidig klargjort til udlæsning.

Det nedenfor viste program foretager en frekvenstælling af tegnene i filen A:LETTER.TXT.

```
PROGRAM tegnfrekvens;
CONST
  bogstaver: SET OF 'A'..'Z' = ['A'..'Z'];
  smaabogst: SET OF 'a'..'z' = ['a'..'z'];
VAR
  antal: ARRAY['A'..'Z'] OF integer;
  ch: char;
  t: TEXT;
BEGIN
  FOR ch:='A' TO 'Z' DO antal[ch]:=0;
  assign(t,'A:LETTER.TXT'); reset(t);
  WHILE NOT eof(t) DO
  BEGIN
    WHILE NOT eoln(t) DO
    BEGIN
      read(t,ch);
      IF ch IN smaabogst THEN ch:=chr(ord(ch)-32);
      IF ch IN bogstaver THEN antal[ch]:=antal[ch]+1;
    END;
    readln(t);
  END;
  close(t);
  writeln('tegn antal');
  FOR ch:='A' TO 'Z' DO
  writeln(' ',ch,antal[ch]:7);
END.
```

read og write standard procedurerne beskrives yderligere i kapitel 16. Her forklares bl.a. hvordan disse procedurer kan bruges til at læse og skrive andre datatyper end tegn.

13.3.2 Logiske I/O enheder

En textfil kan bruges til at kommunikere med MS-DOS eller CP/M operativsystemets logiske I/O enheder. Dette opnås ved at sammen-

knytte textfilen med I/O enhedens symbolske navn via assign proceduren. De følgende I/O enheder er tilgængelige fra PolyPascal:

CON: Console enheden. Udlæsninger sendes til MS–DOS eller CP/M console output enheden (normal systemets skærm), og indlæsninger hentes fra MS–DOS eller CP/M console input enheden (normalt tastaturet). CON: enheden er linieorienteret under indlæsning: Tegnene læses altid fra en liniebuffer, og når denne er tom, bliver en ny linie indlæst. Linierne kan editeres når de indtastes. Afsnit 16.1 indeholder flere detaljer om indlæsning fra CON: enheden.

TRM: Terminal enheden. Refererer til den samme enhed som CON:.

KBD: Keyboard enheden (kun indlæsning). Indlæsninger hentes fra MS–DOS eller CP/M console input enheden (normalt tastaturet). De indlæste tegn udskrives ikke, og indlæsningen er ikke linieorienteret som ved CON: enheden.

LST: List enheden (kun udlæsning). Udlæsninger sendes til MS–DOS eller CP/M list enheden (normalt systemets printer).

AUX: Auxiliary enheden. Udlæsninger sendes til MS–DOS eller CP/M auxiliary enheden, og indlæsninger hentes fra MS–DOS eller CP/M auxiliary enheden. Normalt refererer disse enheder til et modem eller en anden ydre enhed.

USR: User enheden. Udlæsninger sendes til den brugerdefinerbare outputrutine, og indlæsninger hentes fra den brugerdefinerbare inputrutine. Yderligere detaljer om brugerdefinerede I/O rutiner gives i kapitel 22.

INP: Kun MS–DOS versionen. Refererer til MS–DOS standard input filen (file–handle nummer 0). Denne enhed kan også bruges i forbindelse med både almindelige filer og typeløse filer.

OUT: Kun MS–DOS versionen. Refererer til MS–DOS standard output filen (file–handle nummer 1). Denne enhed kan også bruges i forbindelse med både almindelige filer og typeløse filer.

ERR: Kun MS–DOS versionen. Refererer til MS–DOS standard error output filen (file–handle nummer 2). Denne enhed kan også bruges i forbindelse med både almindelige filer og typeløse filer.

MS–DOS brugere bør bemærke, at MS–DOS operativsystemet i sig selv har en række logiske enheder, så som 'CON', 'LST', og 'AUX'. PolyPascal behandler disse enheder som var de diskfiler, dog med en undtagelse: Når man åbner en fil, via reset, rewrite, eller append, undersøger PolyPascal om filen er en logisk enhed (et device) ved hjælp af et MS–DOS systemkald. Hvis MS–DOS rapporterer, at der er tale om en logisk enhed, sætter PolyPascal filens bufferstørrelse til 1, og alle ind– og udlæsninger foregår tegn for tegn. Device–check faciliteten styres ved hjælp af D compilerflaget. Normalstillingen er {$D+}, og i denne stilling foretages device–checks som beskrevet ovenfor. Hvis man placerer et {$D–} direktiv i begyndelsen af programmet (bemærk: før erklæringsdelen), foretager PolyPascal ikke device–checks, og alle I/O operationer foregår som på diskfiler via filens buffer. I så tilfælde kan et kald til flush proceduren bruges til at sikre, at de tegn man har skrevet til en fil faktisk er sendt til den.

Logiske enheder skal, i lighed med diskfiler, åbnes via et kald til reset, rewrite, eller append førend de anvendes. Funktionen af reset, rewrite, og append er ens for logiske enheder. close proceduren har ingen effekt på en logisk enhed.

Det nedenfor viste program udskriver filen B:MESSAGE.TXT på systemets printer:

```
PROGRAM listfil;
VAR
  ch: char;
  indfil,udfil: TEXT;
BEGIN
  assign(indfil,'B:MESSAGE.TXT'); reset(indfil);
  assign(udfil,'LST:'); rewrite(udfil);
  WHILE NOT eof(indfil) DO
  BEGIN
    WHILE NOT eoln(indfil) DO
    BEGIN
      read(indfil,ch); write(udfil,ch);
    END;
```

```
    readln(indfil); writeln(udfil);
  END;
  close(indfil); close(udfil);
END.
```

Standardprocedurerne eof og eoln arbejder ens på diskfiler og logiske enheder. Som tidligere beskrevet returnerer eof sand når den fysiske slutning af filen er nået, eller når det næste tegn i filen er et Ctrl–Z. eoln returnerer sand når det næste tegn i filen er et CR eller når eof er sand. eof og eoln kan således "se forud" i filen. For at kunne se forud er det imidlertid nødvendigt at læse et tegn forud i filen, dog uden at tegnet går tabt. Når eof eller eoln læser forud, gemmer de derfor det læste tegn i en look–ahead buffer, og denne buffer tømmes da af den næste læsning. Dette kan bedst anskueliggøres ved et eksempel:

```
WHILE NOT eof(t) DO
BEGIN
  WHILE NOT eoln(t) DO read(t,ch);
  readln(t);
END;
```

Når eof kaldes, læser den et tegn forud og gemmer dette tegn i look–ahead bufferen. Hvis tegnet ikke er et end–of–file mærke, startes den indre WHILE løkke. Ved det første kald til eoln er der allerede læst forud, så eoln værdien kan umiddelbart udregnes. Hvis eoln er falsk, tømmes look–ahead bufferen af read proceduren. I de følgende gennemløb af den indre WHILE løkke bliver alle indlæsning foretaget af eoln, hvorefter read tømmer look–ahead bufferen. Når den indre WHILE løkke slutter, fjerner readln end–of–line mærket fra look–ahead bufferen, og hele processen kan nu gentages.

13.3.3 Standardfiler

Nedenfor vises en oversigt over standardfilerne i PolyPascal. Ved at anvende disse filer fremfor erklærede filer, er det muligt at spare den plads, der ellers skulle bruges til filvariablene og deres initialisering. Alle filerne er præ–tilskrevne til logiske enheder, og det er ikke nødvendigt at åbne dem førend de anvendes (faktisk er det forbudt at anvende assign, reset, rewrite og close på disse filer).

input Den primære inputfil. Denne fil er refererer normalt til CON: enheden. I MS–DOS versionen refererer den til INP: enheden, hvis input buffer størrelsen er forskellig fra nul (G direktivet). Se afsnit 13.3.4 for flere detaljer.

output Den primære outputfil. Denne fil refererer normalt til CON: enheden. I MS–DOS versionen refererer den til OUT: enheden, hvis output buffer størrelsen er forskellig fra nul (P direktivet). Se afsnit 13.3.4 for flere detaljer.

con Refererer til CON: enheden.

kbd Refererer til KBD: enheden.

lst Refererer til LST: enheden.

aux Refererer til AUX: enheden.

usr Refererer til USR: enheden.

Da standardfilerne input og output anvendes meget ofte, vælger compileren automatisk disse filer hvis intet andet angives. Nedenfor vises en liste af textfiloperationer og deres forkortede versioner:

write(ch)	svarer til	write(output,ch)
read(ch)	svarer til	read(input,ch)
writeln	svarer til	writeln(output)
readln	svarer til	readln(input)
eof	svarer til	eof(input)
eoln	svarer til	eoln(input)

Yderligere udvidelser af read og write procedurerne (til ind– og udlæsning af andre datatyper end tegn) beskrives i kapitel 16.

13.3.4 MS–DOS I/O omdirigering

MS–DOS versionen af PolyPascal–86 supporterer MS–DOS operativsystemets I/O omdirigerings facilitet (I/O redirection). Kort fortalt går I/O omdirigering ud på, at man har mulighed for at ændre standard input enheden og/eller standard output enheden til en diskfil. Desuden gælder der, at når et program supporterer I/O omdirigering, kan det også anvendes som et "filter" gennem en "pipe". Yderligere beskrivelse af I/O omdirigering, filtre og pipes findes i MS–DOS operativsystemets dokumentation.

I/O omdirigering aktiveres ved hjælp af G (get) og P (put) compilerdirektiverne. G direktivet kontrollerer inputfilen og P direktivet kontrollerer outputfilen. Begge direktiver kræver et tal, der definerer størrelsen af input eller outputbufferen. Ved start vælger compileren automatisk {$G0} og {$P0}, og når bufferstørrelserne er nul, refererer input og output standardfilerne til CON: eller TRM: enheden som beskrevet i afsnit 13.3.3. Hvis man definerer inputbufferens størrelse til at være forskellig fra nul, f.eks. {$G256}, vil input standardfilen (input identifieren) i stedet referere til MS–DOS standard input filen. Ligeledes gælder der, at hvis man definerer outputbufferens størrelse til at være forskellig fra nul, f.eks. {$P1024} vil output standardfilen (output identifieren) i stedet referere til MS–DOS standard output filen. D compilerdirektivet (se afsnit 13.3.2) gælder også for input og output filerne når disse refererer til MS–DOS standard I/O filerne. Eventuelle P og G compilerdirektiver skal placeres i begyndelsen af et program, dvs. før erklæringsdelen, for at have den ønskede effekt.

Det nedenfor viste program indlæser linier fra MS–DOS standard input filen, sorterer dem, og udlæser dem til MS–DOS standard output filen. Linierne gemmes på heap'en og refereres til gennem et pointerarray. Bemærk, at allocate proceduren anvendes til at reservere præcis det antal bytes, der behøves for hver linie. Havde man i stedet anvendt new proceduren, var lagerforbruget steget kraftigt. Se mere om new og allocate procedurerne i afsnit 14.2. Sorteringen er baseret på den meget effektive quicksort algoritme.

```
PROGRAM sort; {$G512,P512,D-}
CONST
  maxlinie = 1000;
  maxlaengde = 80;
TYPE
  liniestr = STRING[maxlaengde];
  linieptr = ^liniestr;
VAR
  i,nlin: integer;
  linie: liniestr;
  lptr: ARRAY[1..maxlinie] OF linieptr;

PROCEDURE quicksort(v,h: integer);
VAR
  i,j: integer;
  sptr,tptr: linieptr;
BEGIN
  i:=v; j:=h; sptr:=lptr[(v+h) DIV 2];
  REPEAT
    WHILE lptr[i]^<sptr^ DO i:=succ(i);
    WHILE sptr^<lptr^[j] DO j:=succ(j);
    IF i<=j THEN
    BEGIN
      tptr:=lptr[i]; lptr[i]:=lptr[j]; lptr[j]:=tptr;
      i:=succ(i); j:=pred(j);
    END;
  UNTIL i>j;
  IF l<j THEN quicksort(l,j);
  IF i<r THEN quicksort(i,r);
END;

BEGIN
  nlin:=0;
  WHILE NOT eof DO
  BEGIN
    nlin:=succ(nlin); readln(line);
    allocate(lptr[nlin],len(line)+1);
    lptr[nlin]^:=line;
  END;
  IF n>1 then quicksort(1,nlin);
  FOR i:=1 TO nlin DO writeln(lptr[i]^);
END.
```

Hvis dette sorterings program bliver oversat til filen SORT.COM, da vil MS–DOS kommando linjen:

```
SORT <DATA.TXT >SORTED.TXT
```

læse alle linjer fra DATA.TXT, sortere dem og skrive dem tilbage til SORTED.TXT.

13.4 Typeløse filer

En typeløs fil bruges primært til direkte læsning og skrivning af sektorer fra og til CP/M eller MS–DOS diskfiler. En typeløs fil erklæreres med det reserverede ord FILE og intet andet, for eksempel:

```
VAR
  datafil: FILE;
```

13.4.1 Operationer på typeløse filer

Alle standard filoperationer, undtagen read og write, kan anvendes på typeløse filer. I MS–DOS versionen af PolyPascal–86 kan der, når en typeløs fil åbnes med reset eller rewrite, eventuelt specificeres en recordlængde:

```
reset(f,s)   eller   rewrite(f,s)
```

s er et heltalsudtryk, der angiver recordlængden i bytes. Hvis intet angives, anvendes en recordlængde på 128. Recordlængden er altid 128 bytes i CP/M versionerne af PolyPascal, og syntaksen for kald til reset og rewrite er derfor denne samme som for almindelige filer.

Til ind– og udlæsninger fra og til typeløse filer anvendes to specielle standardprocedurer, kaldet blockread og blockwrite. Formatet af kald til disse procedurer er:

```
blockread(f,v,c,n)   og   blockwrite(f,v,c,n)
```

hvor f er en typeløs fil, v er en vilkårlig variabel, c er et udtryk af typen integer, og n er en heltalsvariabel. Ved et kald til blockread eller blockwrite overføres op til c records fra/til diskfilen til/fra lageret, startende med den første byte, der optages af variablen v. n returnerer det faktiske antal af overførte records. Det er op til programmøren at sikre, at v er stor nok til at indeholde hele den læste/skrevne datamængde. Efter hver læsning/ skrivning af en record, flyttes fil–pointeren frem til den næste record.

Hvis n returnerer med en værdi forskellig fra c ved et kald til blockread, betyder det, at slutningen af filen er nået. Hvis n returnerer med en værdi forskellig fra c ved et kald til blockwrite, betyder det, at disken er fuld.

Ind– og udlæsninger fra og til en typeløs fil skal, i lighed med almindelige filer, foregås af et kald til assign og et kald til reset eller rewrite. Hvis rewrite anvendes, oprettes og åbnes en ny fil. Hvis reset anvendes, åbnes en eksisterende fil. Hvis reset eller rewrite anvendes på en typeløs fil, skal close også kaldes ved endt behandling, for at sikre en korrekt afslutning.

Effekten af et kald til seek proceduren eller en af funktionerne length (longlen), position (longpos) og eof med en typeløs fil, svarer til et tilsvarende kald med en almindelig fil.

Det nedenfor viste program bruger to typeløse fil variabler til at kopiere data fra en disk fil til en anden.

```
PROGRAM kopier;
TYPE
  buffer = ARRAY[0..32767] OF byte;
VAR
  m,n: integer;
  indnavn,udnavn: STRING[64];
  indfil,udfil: FILE;
  buf: buffer;
BEGIN
  write('Kopier fra? '); readln(indnavn);
  write('Kopier til? '); readln(udnavn);
  assign(indfil,indnavn); reset(indfil,1);
  assign(udfil,udnavn); rewrite(udfil,1);
  REPEAT
    blockread(indfil,buf,size(buffer),n);
    IF n<>0 THEN
    BEGIN
      blockwrite(udfil,buf,n,m);
      IF m<>n THEN
      BEGIN
        writeln('Ikke plads i outputfilen'); n:=0;
      END;
    END;
  UNTIL n=0;
  close(indfil); close(udfil);
END.
```

Foranstående eksempel er beregnet til MS-DOS versionen af Poly-Pascal-86. I CP/M versionen skal recordstørrelsen udelades i kaldene til reset og rewrite, og i kaldet til blockread skal "size(buffer)" erstattes med "256", da bufferen netop kan indeholde 256 sektorer, hver på 128 bytes.

13.5 I/O kontrol

{$I+} og {$I-} compilerdirektiverne bestemmer, om compileren skal generere kode til kørselskontrol af I/O operationer. Ved start af compileren bliver {$I+} automatisk valgt, og i denne stilling vil alle I/O operationer blive kontrolleret under kørsel af programmet. I den modsatte stilling, {$I-}, bliver der ikke genereret kontrolkode. Status af en I/O operation kan i stedet kontrolleres ved at udføre et kald til standardfunktionen iores. I Appendix F findes en oversigt over de mulige statuskoder og deres betydning. Bemærk, at sålænge der ikke forekommer fejl, returnerer iores nul. Bemærk også, at i tilfælde af fejl bliver alle I/O operationer suspenderet, indtil iores kaldes. Først når dette sker, forlades fejltilstanden, hvorefter yderligere I/O operationer kan foretages.

iores faciliteten er brugbar i mange situationer. Den nedenfor viste programstump viser, hvorledes iores kan anvendes til at undersøge, om en given diskfil eksisterer eller ej:

```
assign(f,'B:LETTER.TXT');
{$I-} reset(f) {$I+};
IF iores>0 THEN writeln('Filen findes ikke.');
```

Når {$I-} stillingen bruges, bør kald til de følgende procedurer og funktioner kontrolleres med et kald til iores:

append	blockread	blockwrite
chain	chdir	close
erase	execute	flush
gcdir	mkdir	read
readln	rename	reset
rewrite	rmdir	seek
write	writeln	

14. Pointere

Indtil nu har denne manual kun beskrevet statiske variable, dvs. variable der ligger på en faste plader i lageret og som refereres med faste identifiere. En dynamisk variabel er en plads, som man selv kan reservere under udførelsen af et program. Med en sådan plads er der ikke associeret et fast navn, men pladsen refereres i stedet med en pointer. Pointerens "værdi" er da blot startadressen på den dynamiske variabel.

14.1 Pointertyper

En pointertype angives med pointersymbolet (^) efterfulgt af typeidentifieren (bemærk: typeidentifieren, ikke typen) for de dynamiske variable, der kan allokeres og refereres gennem pointervariable af denne type. Nogle eksempler:

```
TYPE
  intptr = ^integer;
  str40 = STRING[40];
  strptr = ^str40;
  data = RECORD
           i: integer; ch: char;
         END;
  dataptr = ^data;
```

Det er tilladt typeidentifieren i definitionen af en pointertype at referere til en endnu ukendt typeidentifier. Bemærk, at dette er det eneste tilfælde, hvor referencer til ukendte identifiere tillades. Et eksempel:

```
TYPE
  link = ^person;
  person = RECORD
             fornavn,efternavn: STRING[32];
             alder: 0..100;
             naeste: link;
           END;
```

Dynamiske variable af den ovenfor viste type person er karakteriserede ved, at de indeholder en pointer til den næste dynamiske variabel i en kædet liste. Pointere anvendes meget ofte til at opbygge sådanne kædede lister.

14.2 Brug af pointere

En dynamisk variabel refereres via en pointer ved at efterfølge pointervariablen med pointersymbolet (^). Antag, at de følgende erklæringer er udført:

```
TYPE
  dims = RECORD
           navn: STRING[20];
           pris: real;
  END;
  dimsptr = ^dims;
  liste = ARRAY[1..5] OF integer;
VAR
  pint: ^integer;
  foerstedims: dimsptr;
  pliste: ^liste;
  i: integer;
```

De følgende tilskrivninger kan da udføres:

```
pint^:=4;
foerstedims^.navn:='knagerække';
foerstedims^.pris:=29.75;
pliste^[3]:=7;
pliste^[2]:=pint^;
pliste^[i]:=pliste^[i+1]*2;
```

En dynamisk variabel allokeres med standardproceduren new. Under forudsætning af de ovenstående erklæringer, vil sætningen:

```
new(foerstedims);
```

altså allokere en dynamisk variabel af typen dims, og tilskrive dens adresse i lageret til pointeren foerstedims.

En pointer kan tilskrives værdien af en anden pointer, forudsat begge pointere er af samme type. To pointere af samme type kan desuden testes for lighed og ulighed med de relationelle operatorer = og <>.

Pointerværdien NIL (bemærk at NIL er et reserveret ord og ikke en standardkonstant) er kompatibel med alle pointertyper. Når en pointer ikke peger på noget objekt, tilskrives den normalt værdien NIL. NIL kan desuden anvendes i sammenligninger af pointere.

Som tidligere nævnt er sammensætning af poster til kædede strukturer en udbredt anvendelse af dynamiske poststrukturer. Hver post skal da indeholde et felt med en pointer til en anden post, normalt af samme type som posten selv. Det nedenfor viste program opbygger en kædet liste af personer, og udskriver derefter listen:

```
PROGRAM liste;
TYPE
  navnetype = STRING[30];
  personptr = ^person;
  person = RECORD
             navn: navnetype;
             naeste: personptr;
           END;
VAR
  personliste,pp: personptr;
  nytnavn: navnetype;
BEGIN
  personliste:=NIL;
  writeln('skriv navne og slut med en blank linie:');
  REPEAT
    readln(nytnavn);
    IF nytnavn<>'' THEN
    BEGIN
      new(pp);
      pp^.navn:=nytnavn;
      pp^.naeste:=personliste;
      personliste:=pp;
    END;
  UNTIL nytnavn='';
  writeln('de følgende navne blev indtastet:');
  pp:=personliste;
  WHILE pp<>NIL DO
  BEGIN
    writeln(pp^.navn); pp:=pp^.naeste;
  END;
END.
```

Dynamiske variable, der oprettes ved kald til new proceduren, lagres på en stak–lignende struktur, der kaldes systemets "heap". PolyPascal styrer heapen gennem en heap–pointer. Ved begyndelsen af et program sættes heap–pointeren til at pege på den første ledige byte i lageret. Hvergang new kaldes, bliver heap–pointeren flyttet opad i lageret svarende til det antal bytes, den nye dynamiske variabel optager.

Hvis en eller flere dynamiske variable af en eller anden grund ikke længere anvendes, kan standardprocedurerne mark og release bruges til at frigive det lager, der optages af disse variable. mark proceduren bruges til at gemme heap–pointerens nuværende adresse i en variabel. Formatet af et kald til mark er:

```
mark(v);
```

hvor v kan være enhver pointervariabel. release proceduren sætter heap–pointeren til den adresse, der er indeholdt i argumentet. Formatet af et kald til release er:

```
release(v);
```

hvor v kan være enhver pointervariabel, der tidligere er sat ved hjælp af mark. Det nedenfor viste program demonstrerer på simpel vis brugen af mark og release.

```
PROGRAM heap;
TYPE
  objekt = RECORD
             navn: STRING[30];
             pris: real;
           END;
VAR
  objektptr: ^objekt;
  heapmaerke: ^integer;
BEGIN
  mark(heapmaerke); new(objektptr);
  objektptr^.navn:='skruetrækker';
  objektptr^.pris:=13.95;
  release(heapmaerke);
END.
```

I begyndelsen af programmet bruges mark til at gemme heap–pointerens startværdi i variablen heapmaerke (typen af heapmaerke er uden betydning, da heapmaerke aldrig bruges i et kald til new). Derefter udføres et kald til new for at allokere en dynamisk variabel af typen objekt, og den dynamiske variabel tilskrives en værdi. Til sidst kaldes release med heapmaerke som argument, hvorved heap–pointeren flyttes tilbage til begyndelsestilstanden, og dermed frigives det lager, der var optaget af den dynamiske variabel.

Hvis der var udført flere kald til new mellem kaldene til mark og release, ville det lager, der var optaget af alle disse dynamiske variable, blive frigivet. På grund af heapens stak–lignende opbygning, er det ikke muligt at frigive en dynamisk variabel midt i heapen. Bemærk desuden, at forkert brug af mark og release kan efterlade "flydende pointere", der peger på dynamiske variable, der ikke længere er en del af den definerede heap.

Hvis man ønsker at allokere et variabelt antal bytes på heapen, kan man bruge standard proceduren allocate. Formatet af et kald til allocate er:

```
allocate(p,n);
```

hvor p kan være enhver pointervariabel og n er et heltalsudtryk, der angiver antallet af bytes, der skal allokeres. p kommer til at pege på det allokerede områdes første byte. Se program eksemplet i afsnit 13.3.4.

Standardfunktionen memavail kan bruges til at undersøge hvormange bytes, der er ledige til yderligere udvidelse af heapen på et givet tidspunkt. memavail tager ingen argumenter, og returnerer en værdi af typen integer. Hvis der er mere en 32767 ledige bytes, returnerer memavail et negativt tal. Den rigtige værdi kan da beregnes af 65536.0–memavail (bemærk at en konstant af typen real anvendes for at gennemtvinge et resultat af denne type – det er nødvendigt da resultatet er større end maxint).

Da PolyPascal–86 tillader en heap, der er større end 64K bytes, er memavail ikke i stand til at rapportere størrelsen af det frie lager i bytes. I stedet returnerer memavail antallet af ledige "paragraffer" (paragraphs). En paragraf svarer til 16 bytes.

Yderligere informationer om systemets brug af lageret findes i kapitel 24.

14.3 Direkte adgang til pointere

I PolyPascal har programmøren direkte adgang til den adresse, der gemmes i en pointer. Dette er uhyre brugbart for en øvet programmør, da man således kan "pege" på enhver adresse i lageret. Hvis det bruges forkert kan det imidlertid også være uhyre farligt, for i uheldige tilfælde kan en dynamisk variabel komme til at ligge oven i en anden variabel, eller, endnu værre, oven i selve programmet.

Formatet af pointere er forskelligt i PolyPascal–80 og PolyPascal–86 (PolyPascal–80 bruger 16–bits pointere hvorimod PolyPascal–86 bruger 32–bits pointere). De metoder, der anvendes for at få direkte adgang til pointere, er derfor også forskellige.

PolyPascal–86
Pointere i PolyPascal–86 er 32–bits værdier. Det er derfor ikke muligt at konvertere en pointer til en integer med ord funktionen, ligesom ptr funktionen ikke kan konvertere en enkelt integer til en pointer. I stedet for ord funktionen, kan ofs og seg funktionerne anvendes til at finde offset– og segmentadresserne på en variabel. For at finde den offsetadresse og den segmentadresse, der er gemt i en pointer, angiver man blot pointervariablen efterfulgt af pointersymbolet, for eksempel:

```
offset:=ofs(p^);
segment:=seg(p^);
```

Pointerværdier frembringes ved hjælp af addr og ptr funktionerne. addr returnerer en pointer til den variabel, der angives som argument, og ptr opbygger en pointerværdi af to heltalsudtryk, for eksempel:

```
p:=ptr(dseg,$80);
```

Det første udtryk angiver segmentadressen og det andet angiver offsetadressen. De pointerværdier, der returneres af ptr, er kompatible med alle pointertyper.

PolyPascal–80
I PolyPascal–80 findes der to standardfunktioner, ord og ptr, der muliggør konvertering mellem pointere og integers. ord returnerer en integer, der svarer til den adresse, der er indeholdt i pointerargumentet, og ptr konverterer en integer til en pointerværdi, der er kompatibel med alle pointertyper.

14.4 Oversigt over pointerrutiner

I PolyPascal findes de følgende procedurer til styring af dynamiske variable:

new(p)	p er en variabel af enhver pointertype. Proceduren allokerer en dynamisk variabel af den type, der er forbundet med p, og gemmer dens adresse i p.

allocate(p,n) p er en variabel af enhver pointer type og n er et heltals udtryk. Proceduren allokerer et area på n bytes på heepen, og indsætter addressen af dette i p.

mark(p) p er en variabel af enhver pointertype. Proceduren gemmer heap–pointerens nuværende værdi i p.

release(p) p er en variabel af enhver pointertype. Proceduren sætter heap–pointeren til den værdi, der er indeholdt i p.

I PolyPascal findes de følgende pointerrelaterede funktioner:

memavail PolyPascal–86: Returnerer antallet af paragraffer mellem heap–pointeren og toppen af det frie lager. Resultatet er af typen integer. Hvis der er mere end 32767 paragraffer ledige, returneres et negativt tal, og den rigtige værdi kan da beregnes af 65536.0–memavail.
PolyPascal–80: Returnerer antallet af bytes mellem heap–pointeren og toppen af det frie lager. Resultatet er af typen integer. Hvis der er mere end 32767 bytes ledige, returneres et negativt tal, og den rigtige værdi kan da beregnes af 65536.0– memavail.

addr(v) Kun PolyPascal–86. Returnerer en pointer til variablen v. Den returnerede værdi er kompatibel med alle pointertyper. Bemærk, at indeksering er tilladt hvis v er et array, ligesom det er tilladt at angive felter hvis v er en poststruktur.

ptr(s,o) Kun PolyPascal–86. Konverterer segment– og offsetadresserne, der angives af heltalsudtrykkene s og o, til en pointerværdi, der er kompatibel med alle pointertyper. Bemærk, at ptr(0,0)=NIL.

ptr(i) Kun PolyPascal–80. Konverterer adressen givet ved integerudtrykket i til en pointerværdi, der er kompatibel med alle pointertyper. Bemærk, at ptr(0)=NIL.

ord(p) Kun PolyPascal–80. Konverterer pointerværdien p til en værdi af typen integer. p kan være af enhver pointertype. Bemærk, at ord(NIL)=0.

Se desuden afsnit 15.3.2.4 der indeholder en række "uklassificerede" funktioner.

15. Procedurer og funktioner

En procedure er en separat programdel, der aktiveres fra en proceduresætning (se afsnit 6.1.2). En funktion er på mange måder lig en procedure, bortset fra, at funktionen beregner og returnerer en værdi, der indgår i det udtryk, hvorfra funktionen aktiveres (se afsnit 5.2).

15.1 Parametre

Procedurer og funktioner (i fællesskab kaldet underprogrammer) kan have parametre, der tillader underprogrammet at gentage dets handlinger med visse variationer. I underprogrammet kan parametrene anvendes på lige fod med almindelige variable.

Parametre, der nævnes i proceduresætningen eller funktionskaldet, kaldes aktuelle parametre. Parametre, der nævnes i underprogrammets erklæring, kaldes formelle parametre. Under kørsel af underprogrammet bliver de formelle parametre erstattet af de aktuelle parametre. Bemærk, at de formelle og de aktuelle parametre skal stemme overens med hensyn til antal, type og rækkefølge. I PolyPascal er der to typer af parametre: Value–parametre og var–parametre. Når symbolet VAR skrives foran en formel parameter i underprogrammets overskrift, er denne parameter en var–parameter, ellers er det en value–parameter.

Hvis en parameter er en value–parameter, oprettes der ved start af underprogrammet en variabel (den formelle parameter), hvis begyndelsesværdi er givet ved den aktuelle parameter. Den aktuelle parameter er et udtryk, hvoraf en variabel er et simpelt tilfælde. Den aktuelle parameter berøres ikke af, at underprogrammet tilskriver nye værdier til den formelle parameter, og value–parametre kan derfor ikke anvendes til at returnere resultater fra underprogrammet.

Hvis en parameter er en var–parameter, er både den aktuelle og den formelle parameter lig den samme lagerplads. I dette tilfælde skal den aktuelle parameter være en variabel. Under kørsel af underprogrammet repræsenteres den aktuelle parameter af den formelle parameter, og tilskrivninger til den formelle parameter vil ændre den aktuelle parameter tilsvarende. Var–parametre kan derfor anvendes til at returnere resultater fra underprogrammet.

Alle adresseberegninger udføres før et kald. Hvis en parameter for eksempel er et element i en arraystruktur, bliver adressen på dette element udregnet før underprogrammet kaldes.

Filvariable skal altid overføres som var–parametre.

Når en stor datastruktur, så som et array, skal overføres til et underprogram, bør en var–parameter i videst muligt omfang anvendes. Dette sparer både tid og plads, da der kun skal overføres et ord, (to ord i PolyPascal–86) der angiver adressen på strukturen, i modsætning til hele strukturen, der overføres hvis en value–parameter anvendes.

15.2 Procedurer

En procedure er enten en standardprocedure eller en selvdefineret procedure. Standardprocedurer behøver, i modsætning til selvdefinerede procedurer, ikke erklæres før de anvendes. Hvis en selvdefineret procedure erklæres med samme navn som en standardprocedure, kan standardproceduren ikke anvendes i den pågældende blok.

15.2.1 Procedureerklæringer

En procedureerklæring består af en overskrift, en erklæringsdel og en sætningsdel.

Overskriften angiver procedurens navn, i form af en identifier, og eventuelt en parameterliste. Parameterlisten består af et antal parametersektioner, adskilt af semikoloner og omsluttet af parenteser. Hver parametersektion består af en eller flere identifiere, adskilt af kommaer, efterfulgt af et kolon og en typeidentifier. Det reserverede ord VAR foran en parametersektion angiver, at parametrene i denne sektion er var–parametre. Nogle eksempler:

```
PROCEDURE afbryd;
PROCEDURE drawto(x,y: integer);
PROCEDURE sorter(VAR navne: liste; fra,til: integer);
```

En procedures erklæringsdel er opbygget på samme måde som et programs. Alle identifiere, der nævnes i procedurens overskrift og i dens erklæringsdel, er lokale for proceduren og andre procedurer og funktioner erklæret inden i denne. Dette kaldes identifierens rækkevidde. En procedure kan referere til identifieren for enhver konstant, type, variabel, procedure eller funktion, der er global i forhold til procedu-

ren (erklæret i en ydre blok), eller den kan i stedet redefinere identifieren.

Sætningsdelen angiver de handlinger, der skal udføres når proceduren kaldes. I lighed med et program er en procedures sætningsdel en sammensat sætning, dvs. et antal sætninger adskilt af semikoloner og omsluttet af de reserverede ord BEGIN og END. Hvis proceduren udfører et kald til sig selv, kaldes den for en rekursiv procedure.

Det nedenfor viste program anvender en procedure med en value–parameter. En value–parameter bruges, fordi den aktuelle parameter i nogle tilfælde er en konstant (et simpelt udtryk), der ikke kan overføres som en var–parameter.

```
PROGRAM histogram;
VAR
  i,k,n: integer;
  tal: real;

PROCEDURE skrivlinie(laengde: integer);
VAR
  i: integer;
BEGIN
  FOR i:=1 TO laengde DO write('*');
  writeln;
END;
BEGIN
  readln(n);
  FOR i:=1 TO n DO
  BEGIN
    readln(tal);
    k:=round(tal);
    IF k<0 THEN skrivlinie(0) ELSE
    IF k>79 THEN skrivlinie(79) ELSE
    skrivlinie(k);
  END;
END.
```

Her er et andet program, der anvender en procedure med to var–parametre. Var–parametre bruges, fordi proceduren skal returnere et resultat, hvilket kun kan gøres via parametre af denne art.

```
PROGRAM sammenlign;
VAR
  a,b: integer;

PROCEDURE ombyt(VAR x,y: integer);
VAR
  temp: integer;
BEGIN
  temp:=x; x:=y; y:=temp;
END;

BEGIN
  readln(a,b);
  IF a=b THEN writeln('Tallene er ens') ELSE
  BEGIN
    IF b>a THEN ombyt(a,b);
    writeln('Det st|rste tal er ',a);
    writeln('Det mindste tal er ',b);
  END;
END.
```

Bemærk, at typerne af en procedures parametre skal specificeres som typeidentifiere. Konstruktionen:

```
PROCEDURE sort(data: ARRAY[1..100] OF integer);
```

er altså ikke tilladt. Den korrekte metode er, at associere en typeidentifier med parametertypen, og derefter at bruge denne identifier i parameterlisten:

```
TYPE
  liste = ARRAY[1..100] OF integer;

PROCEDURE sort(data: liste);
```

15.2.2 Standardprocedurer

Standardprocedurerne til strengbehandling er beskrevet i afsnit 8.4, standardprocedurerne til filbehandling er beskrevet i afsnittene 13.2, 13.3.3, og 13.4.1, standardprocedurerne til styring af dynamiske variable er beskrevet i afsnit 14.4, og standardprocedurerne til ind– og udlæsning er beskrevet i kapitel 16. Herudover findes de følgende standardprocedurer:

exit — Returnerer fra den nuværende blok. Hvis exit kaldes i en procedure eller i en funktion, returneres der fra subrutinen. Hvis exit kaldes i hovedprogrammet, stopper programmet. Et kald til exit svarer faktisk til en GOTO til en imaginær label, der er placeret lige før slutningen af den nuværende blok.

halt — Stopper programmet. Hvis programmet blev startet fra operativsystemet, returneres der hertil. Hvis programmet blev startet fra PolyPascal med en RUN kommando, returneres der til kommandoniveauet.

gotoxy(x,y) — Flytter skærmterminalens markør til linie y position x. x og y er udtryk af typen integer. Skærmens øverste venstre hjørne svarer til 0,0.

randomize — Klargør randomgeneratoren ved at anvende et tilfældigt tal som udgangspunkt. I PolyPascal–80 fås det tilfældige fra Z–80 processorens refresh register (R). I PolyPascal–86 har randomize ingen betydning. Random–generatoren må her initialiseres ved et tilfældigt antal kald af random funktionen.

move(s,d,n) — Foretager en kopiering af en blok i lageret. s og d er variable af enhver type, og n er et udtryk af typen integer. Der kopieres n bytes, fra det lagerområde, der starter med den første byte i s, til det lagerområde, der starter med den første byte i d.

fill(d,n,p) — Udfylder en blok i lageret. d er en variabel af enhver type, n er et udtryk af typen integer, og p er et udtryk af typen byte eller af typen char. Værdien p fyldes ind i n på hinanden følgende bytes, startende med den første byte, der optages af d.

15.3 Funktioner

En funktion er, i lighed med en procedure, enten en standardfunktion eller en selvdefineret funktion.

15.3.1 Funktionserklæringer

En funktionserklæring består af en overskrift, en erklæringsdel og en sætningsdel.

En funktions overskrift er magen til en procedures, bortset fra, at parameterlisten skal efterfølges af et kolon og en typeidentifier, der bestemmer typen af den værdi, funktionen returnerer. Nogle eksempler:

```
FUNCTION klar: boolean;
FUNCTION konverter(s: str): integer;
FUNCTION max2(a,b: real): real;
```

Resultattypen skal enten være en skalar type (integer, real, boolean, char, selvdefineret skalar eller delinterval), en pointertype eller en strengtype.

En funktions erklæringsdel er magen til en procedures.

Sætningsdelen er en sammensat sætning, dvs. et antal sætninger adskilt af semikoloner og omsluttet af de reserverede ord BEGIN og END. I sætningsdelen skal der være mindst en sætning, der tilskriver en værdi til funktionsidentifieren. Denne tilskrivning bestemmer funktionens resultat. Hvis funktionen udfører et kald til sig selv, kaldes den for en rekursiv funktion.

Det nedenfor viste program anvender en funktion til at finde den største af fire værdier:

```
PROGRAM findmax;
VAR
  i,j,k,l: integer;

FUNCTION max4(a,b,c,d: integer): integer;

FUNCTION max2(a,b: integer): integer;
BEGIN
  IF a>b THEN max2:=a ELSE max2:=b;
END;
```

```
BEGIN
  max4:=max2(max2(a,b),max2(c,d));
END;

BEGIN
  readln(i,j,k,l);
  writeln('Det st|rste tal er ',max4(i,j,k,l));
END.
```

Bemærk, at funktionen max2 i det ovenstående eksempel er lokal indenfor den blok, der udgøres af max4 funktionen, og at den derfor ikke kan kaldes fra hovedprogrammet.

Det nedenfor viste program demonstrerer anvendelsen af en rekursiv funktion til at beregne fakulteten af et heltal:

```
PROGRAM beregnfak;
VAR
  n: integer;

FUNCTION fak(n: integer): integer;
BEGIN
  IF n<=1 THEN fak:=1 ELSE fak:=n*fak(n-1);
END;

BEGIN
  readln(n);
  writeln(n,'! = ',fak(n));
END.
```

Bemærk, at en funktions resultattype skal specificeres som en typeidentifier. Konstruktionen:

```
FUNCTION hex(tal,cifre: integer): STRING[4];
```

er derfor ikke tilladt. I stedet bør en typeidentifier associeres med typen STRING(.4.), og denne identifier kan derefter bruges til at angive funktionens resultattype:

```
TYPE
  hexstr = STRING[4];

FUNCTION hex(tal,cifre: integer): hexstr;
```

Hvis en funktion kalder en eller flere af procedurerne read, readln, write, og writeln, må denne funktion aldrig aktiveres fra et udtryk i en write eller writeln sætning. Grunden til dette ligger i måden, hvorpå write og writeln er implementeret. Ved starten af et kald til en af disse procedurer sættes visse interne variable til nogle værdier, der skal gælde for hele kaldet. Hvis en funktion, der kaldes fra et udtryk i en write eller writeln sætning, derefter udfører endnu et kald, vil dette kald ødelægge informationerne for det "ydre" kald.

15.3.2 Standardfunktioner

Standardfunktionerne til strengbehandling er beskrevet i afsnit 8.4, standardfunktionerne til filbehandling er beskrevet i afsnittene 13.2 og 13.3.1, og standardfunktioner, der kan relateres til pointere, er beskrevet i afsnit 14.4.

15.3.2.1 Aritmetiske funktioner

I de nedenfor viste funktioner er typen af x enten real eller integer og typen af resultatet den samme som x.

abs(x)	Den absolutte værdi af x.
sqr(x)	x kvadreret (x*x).

I de nedenfor viste funktioner er typen af x enten real eller integer og typen af resultatet real.

sin(x)	Sinus til x.
cos(x)	Cosinus til x.
arctan(x)	Arccus tangens til x.
ln(x)	Den naturlige logaritme til x.
exp(x)	Grundtallet e opløftet i x'ene potens.
sqrt(x)	Kvadratroden af x.
int(x)	Heltalsdelen af x, dvs. det største hele tal, der er mindre end eller lig med x, for x>=0, eller det mindste hele tal, der er større end eller lig med x, for x<0.

frac(x) Decimaldelen af x med samme fortegn som x. Beregnes af frac(x)=x–int(x).

Funktionerne sin, cos, arctan, ln, exp og sqrt er ikke tilgængelige i Business versionen af PolyPascal.

15.3.2.2 Skalare funktioner

succ(x) x er af en skalar datatype, og resultatet er efterfølgeren til x (hvis en sådan findes).

pred(x) x er af en skalar datatype, og resultatet er forgængeren til x (hvis en sådan findes).

odd(x) x er af typen integer. Returnerer sand (true) hvis x er ulige eller falsk (false) hvis x er lige.

15.3.2.3 Konverteringsfunktioner

trunc(x) x er af typen real, og resultatet er det største heltal, der er mindre end eller lig med x, for x>=0, eller det mindste heltal, der er større end eller lig med x, for x<0. Resultatet er af typen integer.

round(x) x er af typen real, og resultatet, der er af typen integer, er den afrundede heltalsværdi af x:

round(x) = trunc(x+0.5), for x>=0
round(x) = trunc(x–0.5), for x<0

ord(x) x kan være af enhver skalar datatype, undtagen real. Resultatet, der er af typen integer, er den ordinale værdi af x.

chr(x) x er af typen integer, og resultatet er et tegn (type char), hvis ordinale værdi (ASCII værdi) er x.

15.3.2.4 Andre standardfunktioner

pwrten(i) i er et heltalsudtryk i området –37 til 37 (–63 til 63 for Business versionen, og –307 til 307 for 8087 versionen af PolyPascal–86). Resultatet, der er af typen real, er 10 opløftet til i'ende potens.

random	Returnerer et tilfældigt tal i området 0.0<=r<1.0. Resultatet er af typen real.
random(i)	Returnerer et tilfældigt heltal i området 0<=r<i. Resultatet er af typen integer.
keypress	Returnerer sand (true) hvis en tast holdes nede på skærmterminalens tastatur, eller falsk (false) hvis ingen nøgler er aktiverede. Tastaturets status undersøges ved at udføre et kald til CP/M eller MS–DOS console status rutinen.
hi(i)	i er af typen integer, og resultatet er af typen integer. Den returnerede værdi er den mest betydende byte af i flyttet til den mindst betydende byte. Den mest betydende byte i resultatet er nul.
lo(i)	i er af typen integer, og resultatet er af typen integer. Den returnerede værdi er den mindst betydende byte af i. Den mest betydende byte i resultatet er nul.
swap(i)	i er af typen integer, og resultatet er af typen integer. Den returnerede værdi findes ved at ombytte den mindst betydende byte og den mest betydende byte.
size(v)	v kan være enhver variabel (se ovenfor) eller en typeidentifier. Den returnerede værdi, der er af typen integer, er størrelsen af v i bytes.
addr(v)	Kun PolyPascal–80. v kan være enhver variabel eller en procedure– eller funktionsidentifier. Den returnerede værdi, der er af typen integer, er v's adresse i lageret. Bemærk, at hvis v er en arraystruktur, er det tilladt at angive indexudtryk, og hvis v er en post, er det tilladt at udvælge felter.

ofs(v) Kun PolyPascal–86. v kan være enhver variabel eller en procedure– eller funktionsidentifier. Funktionen returnerer v's offsetadresse (procedurer og funktioner ligger altid i kodesegmentet, hvis segmentadresse fås ved et kald af cseg funktionen). Bemærk, at indexering er tilladt, hvis v er et array, ligesom det er tilladt at angive felter, hvis v er en poststruktur. Den returnerede værdi er af typen integer.

seg(v) Kun PolyPascal–86. Returnerer segmentadressen på variablen v. Bemærk, at indexering er tilladt, hvis v er et array, ligesom det er tilladt at angive felter, hvis v er en poststruktur. Den returnerede værdi er af typen integer.

cseg Kun PolyPascal–86. Returnerer segmentadressen på kodesegmentet. Den returnerede værdi er af typen integer.

dseg Kun PolyPascal–86. Returnerer segmentadressen på datasegmentet. Den returnerede værdi er af typen integer.

sseg Kun PolyPascal–86. Returnerer segmentadressen på staksegmentet. Den returnerede værdi er af typen integer.

15.4 FORWARD specifikationer

Ved hjælp af FORWARD specifikationer er det muligt at referere til et underprogram førend den faktiske definition af underprogrammet er givet. Dette bliver nødvendigt, hvis to underprogrammer er indbyrdes rekursive, altså hvis det ene underprogram kalder det andet, der igen kalder det første, da det er umuligt at definere begge underprogrammer fuldtud, førend de kaldes. En FORWARD erklæring foretages ved at separere underprogrammets overskrift fra dets programblok. Overskriften (og den fuldstændige parameterliste) skrives først, efterfulgt af det reserverede ord FORWARD, og selve blokken følger senere i den samme erklæringsdel. Bemærk, at parameterlisten ikke skal gentages, når programblokken defineres. Det nedenfor viste program demonstrerer brugen af en FORWARD specifikation.

```
PROGRAM flipflop;

PROCEDURE flip(n: integer); FORWARD;

PROCEDURE flop(n: integer);
BEGIN
  write('indgang til flop. n=',n);
  IF n>0 THEN flip(n-1);
  write('udgang fra flop. n=',n);
END;

PROCEDURE flip;
BEGIN
  write('indgang til flip. n=',n);
  IF n>0 THEN flop(n-1);
  write('udgang fra flip. n=',n);
END;

BEGIN
  flip(3);
END.
```

Programmet giver følgende udskrift:

```
indgang til flip. n=3
indgang til flop. n=2
indgang til flip. n=1
indgang til flop. n=0
udgang fra flop. n=0
udgang fra flip. n=1
udgang fra flop. n=2
udgang fra flip. n=3
```

15.5 Strenge som var-parametre

For var-parametre (parametre, der er erklæret med VAR) gælder der, at den formelle parameter og den aktuelle parameter skal være af en og samme datatype. Dette betyder normalt, at procedurer og funktioner med strenge som var-parametre kun kan anvendes på en bestemt strengtype (dvs. strenge med en bestemt maksimal længde), hvilket ofte er irriterende. Ved hjælp af et {$V-} compilerdirektiv er det derfor muligt, at instruere compileren om, at tillade enhver strengtype som aktuel var-parameter, selvom dens maksimale længde ikke stem-

mer overens med den formelle var–parameter. Ved start af compileren vælges {$V+}, og i denne stilling skal typerne være ens. Et eksempel:

```
TYPE
  streng: STRING[255];
VAR
  kort: STRING[16]; lang: STRING[64];

PROCEDURE storebogst(VAR s: streng);
VAR
  i: integer;
BEGIN
  FOR i:=1 TO len(s) DO
  IF s[i] IN ['a'..'z'] THEN s[i]:=chr(ord(s[i])–32);
END;

BEGIN {$V–}
  readln(kort); storebogst(kort); writeln(kort);
  readln(lang); storebogst(lang); writeln(lang);
END.
```

15.6 Typeløse var–parametre

Til visse specielle formål er det en fordel at kunne skrive procedurer og funktioner, der accepterer alle variable, uanset deres type, som var–parametre. Fænomenet kendes for eksempel fra move, fill, blockread, og blockwrite standard procedurerne, hvor visse af parametrene kan være variable af enhver type. Ved at undlade at angive en datatype for en var–parameter i et underprograms overskrift, opnås, at denne parameter er typeløs, og dermed at den aktuelle parameter kan være enhver variabel. En typeløs parameter er inkompatibel med alle datatyper, og den kan derfor kun bruges de steder, hvor datatypen er uden betydning, så som parameter til move, fill, blockread, blockwrite og addr standard rutinerne, samt som adresseangiver i en AT specifikation.

Den nedenfor viste funktion demonstrerer brugen af typeløse parametre. Funktionen beregner gennemsnittet af en angiven del af et heltalsarray.

```
FUNCTION gennemsnit(VAR liste; antal: integer): real;
VAR
  data: ARRAY[0..maxint] OF integer AT liste;
  i: integer;
  sum: real;
BEGIN
  sum:=0;
  FOR i:=0 TO pred(antal) DO sum:=sum+data[i];
  gennemsnit:=sum/antal;
END;
```

Hvis den følgende erklæring er foretaget:

```
VAR
  talliste: ARRAY[1..100] OF integer;
```

da kan gennemsnit anvendes som vist nedenfor:

```
writeln('Gennemsnit(1..100) = ',gennemsnit(talliste,100));

FOR i:=0 TO 9 DO
BEGIN
  position=i*10+1;
  write('Gennemsnit(',position,'..',position+9,') = ');
  writeln(gennemsnit(talliste[position],10):0:4);
END;
```

15.7 Absolutte procedurer og funktioner

Dette afsnit gælder kun PolyPascal–80. Normalt tillader PolyPascal–80, at procedurer og funktioner er rekursive. Imidlertid bruges denne facilitet sjældent, og det er derfor muligt, ved hjælp af {$A+} og {$A–} compilerdirektiverne, at instruere compileren om at generere absolut kode for underprogrammer. Absolut kode er både hurtigere og mere kompakt end rekursiv kode. Ved start af compileren vælges {$A–} automatisk, og underprogrammer, der oversættes i denne stilling, tillader rekursion. I den modsatte stilling, {$A+}, genereres absolut kode. Absolutte procedurer og funktioner fungerer kun korrekt, hvis de nedenstående krav er opfyldt:

Procedurens eller funktionens identifier må ikke forekomme i en proceduresætning eller i et udtryk i underprogrammets sætningsdel (direkte rekursion).

Procedurer eller funktioner, der udfører kald til underprogrammet, må ikke kaldes fra underprogrammet (indirekte rekursion).

I PolyPascal–86 kan procedurer og funktioner altid udføres rekursivt, og A compilerdirektivet har derfor ingen betydning i denne version. Af hensyn til kompatibilitet med PolyPascal–80 tillades det dog.

15.8 Stak overflow kontrol

I PolyPascal–86 findes et K compilerdirektiv, der styrer compilerens generering af kode til stak overflow kontrol før kald af procedurer og funktioner. Ved start af compileren vælges {$K+} automatisk, og i denne stilling genererer compileren kode til stak overflow kontrol. I {$K–} stillingen genereres denne kode ikke. K compilerdirektivet har igen betydning i PolyPascal–80, men det tillades dog af hensyn til kompatibilitet med PolyPascal–86.

15.9 Overlay procedurer og funktioner

I PolyPascal er der mulighed for at adskille grupper af procedurer og funktioner fra hovedprogrammet. Dette er en stor fordel når man udvikler store programmer, idet sådanne programmer fylder mindre i lageret når de udføres. Dog er de alligevel at betragte som enkelte programmer, og ikke flere kædede (chainede) programmer.

Alle procedurer og funktioner, der erklæres med det reserverede ord OVERLAY, bliver under oversættelsen adskilt fra hovedprogrammet og placeret i en eller flere separate overlay–filer. Under kørsel af programmet bliver overlay–underprogrammerne kun indlæst fra disken, når de kaldes. Da flere overlay–underprogrammer kan dele det samme lagerområde i hovedprogrammet, dvs. indlæses og udføres på samme adresse i lageret, kan man nu have adskillige procedurer og/eller funktioner i et program, der kun optager plads for en (nemlig den største).

Det er her vigtigt at lægge mærke til forskellen mellem overlay–underprogrammer og kædede (chainede) programmer: Overlay–underprogrammer oversættes sammen med hovedprogrammet, og udgør således et hele, mens kædede programmer oversættes separat.

I PolyPascal–80 kan overlay–underprogrammer ikke anvendes i programmer, der startes med RUN kommandoen. Overlay–underprogrammer fungerer kun i programfiler, altså programmer, der er oversat med PROGRAM eller OBJECT kommandoerne, og startes fra CP/M.

Erklæring af overlay–underprogrammer foregår på samme måde som erklæring af almindelige underprogrammer, bortset fra, at erklæringen startes med det reserverede ord OVERLAY. Eksempler:

```
OVERLAY PROCEDURE initialize;
VAR
  i: integer;
BEGIN
  FOR i:=1 TO 10 DO data[i]:=0;
  count:=0;
END;

OVERLAY FUNCTION average(d: list): real;
VAR
  i: integer;
  a: real;
BEGIN
  a:=0;
  FOR i:=1 TO 25 DO a:=a+d[i];
  average:=a/25;
END;
```

Når et program oversættes, bliver overlay–underprogrammer, der står umiddelbart efter hinanden, samlet i en enkelt overlay–fil. I hovedprogrammets maskinkode bliver der derefter afsat et overlay–område, der er stort nok til det største af overlay–underprogrammerne. Et eksempel:

```
PROGRAM overlaydemo_1;

OVERLAY PROCEDURE p1;
BEGIN writeln('procedure 1'); END;

OVERLAY PROCEDURE p2;
BEGIN writeln('procedure 2'); END;

OVERLAY PROCEDURE p3;
BEGIN writeln('procedure 3'); END;

BEGIN p1; p2; p3; END.
```

Antag, at det ovenfor viste program oversættes med PROGRAM kommandoen og gives navnet OVDEMO. Compileren producerer da to filer, nemlig OVDEMO.COM og OVDEMO.000. OVDEMO.COM indeholder hovedprogrammet, og dermed også det område hvori overlay–underprogrammerne indlæses. OVDEMO.000 indeholder maskinkoden for procedurerne p1, p2 og p3. Når OVDEMO.COM udføres, bliver p1, p2 og p3 på skift læst ind i lageret fra OVDEMO.000 og udført.

Ved et kald af et overlay–underprogram undersøger systemet først, om underprogrammet allerede er tilstede i overlay–området. I så fald udføres underprogrammet med det samme, uden læsning fra disken. Hvis hovedprogrammet i eksemplet ovenfor havde udført fem kald til proceduren p1, uden at kalde p2 eller p3 ind i mellem, ville p1 altså kun blive indlæst ved det første kald.

Bemærk, at overlay–underprogrammer, der ligger i samme overlay–fil, *ikke* må kalde hinanden indbyrdes. For det ovenfor viste program gælder der altså, at p1, p2 og p3 *ikke* må kalde hinanden. De kan kun kaldes fra hovedprogrammet eller fra andre underprogrammer.

Et program er ikke begrænset til kun at have en overlay–fil. Faktisk kan der være op til 100 overlay–filer (nummereret fra 000 til 099) knyttet til et enkelt program. Hver overlay–fil har da et bestemt over-

lay–område i hovedprogrammets kode, hvori underprogrammerne indlæses. På denne måde kan flere overlay–underprogrammer være tilstede i lageret på samme tid. Som nævnt ovenfor bliver overlay–underprogrammer, der står umiddelbart efter hinanden, samlet i en enkelt overlay–fil. Hvis der derimod foretages andre erklæringer mellem overlay–underprogrammerne, produceres der flere filer, og dermed flere overlay–områder. Et eksempel:

```
PROGRAM overlaydemo_2;

OVERLAY PROCEDURE p1;
BEGIN END;

OVERLAY PROCEDURE p2;
BEGIN END;

PROCEDURE p3;
BEGIN END;

OVERLAY PROCEDURE p4;
BEGIN END;

OVERLAY PROCEDURE p5;
BEGIN END;

BEGIN END.
```

Hvis det ovenfor viste program oversættes med navnet OVDEMO, producerer compileren tre filer, nemlig OVDEMO.COM, OVDEMO.000 og OVDEMO.001. OVDEMO.000 indeholder koden for p1 og p2, OVDEMO.001 indeholder koden for p4 og p5, og OVDEMO.COM indeholder koden for p3 og hovedprogrammet, samt to overlay–områder. Under kørsel af programmet kan p1 eller p2 være tilstede i lageret samtidig med p4 eller p5. Bemærk, at den erklæring (eller de erklæringer), der adskiller de to overlay–grupper, ikke nødvendigvis skal være et underprogram; det kunne lige så godt være en LABEL, CONST eller VAR erklæring, men en kommentar er ikke nok, da denne helt udelukkes under compileringen.

Som det er tilfældet med almindelige underprogrammer, kan overlay–underprogrammer også nestes (overlays inden i overlays). Et eksempel:

```
PROGRAM overlaydemo_3;

OVERLAY PROCEDURE p1;
BEGIN END;

OVERLAY PROCEDURE p2;

OVERLAY PROCEDURE p21;
BEGIN END;

OVERLAY PROCEDURE p22;
BEGIN END;

BEGIN p21; p22; END;

OVERLAY PROCEDURE p3;
BEGIN END;

BEGIN p1; p2; p3; END.
```

Compileren vil da producere to overlay–filer. 000–filen indeholder koden for p1, p2 og p3, og 001–filen indeholder koden for p21 og p22. I lighed med hovedprogrammet, er der i proceduren p2 afsat et overlay–område, hvori p21 og p22 kan indlæses.

Da overlay–underprogrammer i samme overlay–fil ikke kan kalde hinanden, vil compileren automatisk lade dem dele det samme dataareal. Overlays sparer altså ikke kun plads i programkoden men også dataområde.

Når compileren oversætter et program med overlays, placerer den overlay–filerne samme sted som hovedprogrammet, dvs på samme disk, og for MS–DOS versionen, i samme bibliotek.

Under kørsel af et program med overlays antager systemet, at overlay–filerne ligger på den nuværende disk (den disk der vælges, når intet angives) og, i MS–DOS versionen, i det nuværende bibliotek. I MS–DOS versionen kan dette ændres ved et kald til ovpath standard rutinen. Denne kaldes således:

```
ovpath(s)
```

hvor s er et streng udtryk der specificerer vejen til et underbibliotek (path). En tilsvarende rutine eksisterer i CP/M versionen, bortset fra at der kun kan specificeres et disk navn:

```
ovdrive(d)
```

hvor d et heltals udtryk. 0 betyder samme disk, 1 betyder disk A, 2 betyder disk B, osv.

Når et program efter opstart udfører det første kald til et af overlay–underprogrammerne i en overlay–fil, bliver filen åbnet af systemet, koden bliver indlæst, hvorefter filen igen lukkes.

Ved anvendelsen af overlay–underprogrammer opnås store besparelser først, når underprogrammerne er relativt store og kaldes relativt sjældent. Desuden gælder der, at så mange underprogrammer som muligt bør samles i hver overlay–fil.

Hvis en kørselsfejl forekommer i en overlay–subrutine, er det ikke altid sikkert, at PolyPascal systemets FIND kommando finder det rigtige sted i kildeteksten. Dette skyldes måden hvorpå FIND virker: FIND starter compileren, der oversætter indtil den møder den adresse, der blev specificeret. Når dette sker, vil compilerens interne kildetekstpointer være placeret ved den fejlramte sætning. I en overlay–fil er der imidlertid flere overlay–subrutiner, der deler det samme adresseområde i programkoden, og dermed flere sætninger, der ligger på den samme adresse. Da FIND kommandoen altid stopper første gang, den når den specificerede adresse, vil FIND normalt finde en sætning i den første overlay–subrutine, selvom fejlen måske er opstået i en af de følgende overlay–subrutiner.

Problemet med FIND i forbindelse med overlay–subrutiner kan kun løses ved, at man selv finder ud af hvilken overlay–subrutine fejlen opstod i, og derefter ændrer programmets kildetekst, så denne rutine er den første i den pågældende overlay–fil. FIND vil da virke korrekt.

15.10 EXTERNAL specifikationer

EXTERNAL specifikationen bruges til at erklære eksterne underprogrammer, almindeligvis underprogrammer skrevet i andre sprog, for eksempel maskinkode. Et eksternt underprogram har ingen programblok (dvs. ingen erklæringsdel og ingen sætningsdel). Det eneste, der angives, er underprogrammets overskrift, efterfulgt af en EXTER-

NAL specifikation. EXTERNAL specifikationens syntaks afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
I PolyPascal–86 skal det reserverede ord EXTERNAL efterfølges af en strengkonstant, der angiver navnet på en diskfil. PolyPascal compileren vil da 'linke' koden i denne fil ind i programmet, når det oversættes. Nogle eksempler:

```
PROCEDURE quicksort(VAR d: navneliste); EXTERNAL 'QSORT';
PROCEDURE point(x,y: integer): boolean; EXTERNAL 'POINT';
```

I CP/M–86 versionen af PolyPascal vælges filtypen '.CMD' hvis brugeren undlader at specificere den. PolyPascal læser alle bytes i CMD–filens CODE segment ind i programkoden. Alle andre segmenter ignoreres. I MS–DOS versionen af PolyPascal vælges filtypen '.COM' hvis brugeren undlader at specificere den. PolyPascal læser alle bytes fra COM–filen ind i programkoden.

Eksterne underprogrammer kodes almindeligvis i 8086 assembler og oversættes derefter til binær kode ved hjælp af operativsystemets assembler–oversætter og linker (ASM86 og GENCMD i CP/M–86 eller MASM og LINK i MS–DOS).

Indlæsningen af et eksternt underprograms kode omfatter ikke en relokering – det er blot en direkte overførsel af data fra den eksterne fil til programkoden. Det er således op til programmøren at sikre, at det eksterne underprograms kode er positions–uafhængig. Alle referencer til adresser i kodesegmentet skal altså være relative (JMP og CALL instruktioner er automatisk relative). Desuden må man ikke referere til adresser i datasegmentet.

Externe filer kan indeholde kode for mere end en enkelt rutine. I så tilfælde erklæres den første rutine som vist ovenfor, medens resten erklæres ved at angive den førstes identifier efterfulgt af en heltalskonstant i kantede paranteser. Heltalskonstanten angiver adressen på den pågældende rutine i forhold til starten af den eksterne fil, dvs. den resulterende adresse er adressen af identifieren plus konstanten. Et par eksempler.

```
PROCEDURE auxio; EXTERNAL 'AUXIO.BIN';
FUNCTION auxready: boolean; EXTERNAL auxio[3];
FUNCTION auxin: char; EXTERNAL auxio[6];
PROCEDURE auxout(ch: char); EXTERNAL auxio[9];
```

Ovenstående eksempel læser filen 'AUXIO.BIN' ind i programkoden, og definerer fire rutiner kaldet auxio, auxready, auxin og auxout med indgange i startadressen for den externe kode plus 0, 3, 6 og 9 respektivt. Hvis en extern fil indeholder flere rutiner, er det en god ide at lade den første del af koden være en hoptabel, da man så kan ændre rutinerne uden at ændre definitionerne.

Kapitel 23 beskriver formatet af parametre, der overføres til eksterne underprogrammer, samt de metoder, der bruges til at referere til parametrene.

En ekstern subrutine skal bevare registrene BP, CS, DS og SS, og den skal returnere ved hjælp af en RET (kort) instruktion.

PolyPascal–80
I PolyPascal–80 består en EXTERNAL specifikation af det reserverede ord EXTERNAL efterfulgt af en heltalskonstant, der definerer adressen på underprogrammet. Eksterne underprogrammer kan have parametre, og formatet af kald til eksterne underprogrammer svarer fuldstændigt til kald af almindelige procedurer og funktioner. Nogle eksempler på erklæringer af eksterne underprogrammer:

```
PROCEDURE moveto(x,y: integer); EXTERNAL $F000;
PROCEDURE drawto(x,y: integer); EXTERNAL $F003;
FUNCTION point(x,y: integer): boolean; EXTERNAL $F006;
PROCEDURE quicksort(VAR d: navneliste); EXTERNAL $1D00;
```

Det er op til programmørens at sikre, at der vitterligt findes et underprogram på den nævnte adresse. I afsnit 23.3 findes yderligere informationer om eksterne underprogrammer og parameteroverførsler.

16. Indlæsning og udlæsning

Indlæsning og udlæsning i læsbar form foretages via en textfil (se afsnit 13.3), der er sammenknyttet med en diskfil eller en af de logiske I/O enheder. For at lette ind- og udlæsninger, kan standardprocedurerne read, readln, write og writeln anvendes med en speciel syntax, der bl.a. tillader et variabelt antal parametre i et kald. Desuden behøver parametrene ikke nødvendigvis være af typen char (som beskrevet i afsnit 13.3) – de kan også være strenge og numeriske udtryk af typerne integer og real. I de sidstnævnte tilfælde foretages der automatisk en konvertering fra eller til en numerisk streng. Hvis den første parameter i et kald til read, readln, write eller writeln er en textfilvariabel, bliver ind- eller udlæsningen foretaget via denne textfil. I modsat fald foretages ind- eller udlæsningen via input eller output standardfilen (se afsnit 13.3.3).

16.1 read proceduren

read proceduren anvendes til at indlæse tegn, strenge og tal. Formatet af proceduresætningen er:

read(v1,v2,...,vn) eller read(f,v1,v2,...,vn)

hvor v1,v2,...,vn angiver variable af typerne char, string, integer eller real. I det første tilfælde indlæses variablene fra skærmterminalen (standardfilen input) og i det andet tilfælde fra textfilen f. Husk, at f skal sammenknyttes med en diskfil eller en I/O enhed, ved hjælp af assign proceduren, og åbnes, ved hjælp af reset proceduren, førend der indlæses værdier fra den.

For en char variabel gælder der, at read læser et enkelt tegn fra filen og gemmer det i variablen. Det næste kald til read vil starte med tegnet, der følger lige efter det netop læste.

For en streng gælder der, at read læser så mange tegn som muligt ind i strengen, undtagen hvis et linieskift mødes, eller hvis filen slutter. Det maksimale antal læste tegn er givet ved strengens maksimale længde. Der gøres ikke forskel på blanktegn og andre tegn. Det næste kald til read vil starte med tegnet, der følger lige efter den netop læste streng, eller med det CR eller Ctrl–Z tegn, der afsluttede linien.

Ved indlæsning af en numerisk værdi (integer eller real) forventer read en tekststreng, der følger reglerne for numeriske konstanter, som beskrevet i afsnit 2.2. Blanktegn, tab–tegn (HT) og linieskift (CR og LF tegn) foran tallet ignoreres. Talstrengen må ikke være længere end 30 tegn, og den skal efterfølges af et blanktegn, et HT tegn, et CR tegn, eller et Ctrl–Z tegn. Hvis det numeriske format ikke er korrekt, gives der en I/O fejl. Ellers konverteres talstrengen til en værdi af den givne type, og værdien gemmes derefter i variablen. Det næste kald til read vil starte med tegnet umiddelbart efter det sidst læste tegn i den numeriske streng.

Et specialtilfælde af numerisk indlæsning er, når eof er sand før indlæsningen startes. I stedet for at tilskrive en ny værdi til variablen, lader read den gamle værdi blive stående.

Indlæsning fra standard filen input eller fra en fil, der refererer til CON: enheden, er linieorienteret. PolyPascal læser altid en linie ad gangen ind i en buffer, og selve læsningen af variablene foretages derefter fra bufferen. Når bufferen er tom, læses en ny linie fra konsollen.

Følgende editeringsnøgler kan anvendes under indlæsning af en linie:

BS	Sletter det sidst indtastede tegn. Denne funktion fin des normalt på tastaturet som BS, BACK, BACKSPACE eller en venstrepil, men den kan altid frembringes ved at taste Ctrl–H.
ESC	Sletter hele linien.
Ctrl	Udfører samme funktion som ESC.
Ctrl–D	Genkalder et tegn fra den sidst indtastede linie.
Ctrl–R	Genkalder hele den sidst indtastede linie.
RETURN	Afslutter indtastningen og gemmer et linieslutmærke i liniebufferen (en CR/LF sekvens). Denne funktion findes normalt på tastaturet som RETURN, ENTER eller NEWLINE.
Ctrl–Z	Afslutter indtastningen og gemmer et filslutmærke i liniebufferen (et Ctrl–Z tegn).

Når read bruges i sin korte form, dvs. når man ikke angiver en fil, bliver data altid læst fra skærmterminalen (CON: enheden). I sin korte form læser read desuden altid en ny linie, selvom der stadig er "ulæste" tegn i liniebufferen, og indlæsningen skal altid afsluttes med RETURN (Ctrl–Z ignoreres). Det afsluttende RETURN udskrives ikke, og linien bliver internt afsluttet med et Ctrl–Z tegn. Hvis der således gives færre værdier i inputlinien end i parameterlisten, bliver overskydende char variable sat til Ctrl–Z, overskydende strenge bliver tomme, og overskydende numeriske variable forbliver uændrede.
B compilerdirektivet bruges til at kontrollere den "tvungne læsning", der beskrives ovenfor. Normaltilstanden er {$B+}, og i denne tilstand vil kald til read uden en filvariabel altid indlæse en ny linie. Hvis et {$B–} direktiv placeres i begyndelsen af programmet (bemærk: før erklæringsdelen), vil den korte form af read opføre sig på samme måde som hvis man havde angivet input standardfilen, dvs:

```
read(v1,v2,...,vn)   svarer til   read(input,v1,v2,...,vn)
```

I dette tilfælde bliver nye linier først indlæst når liniebufferen er tom. {$B–} tilstanden følger definitionen af Standard Pascal I/O, medens {$B+} tilstanden giver bedre kontrol med indlæsningerne.

Normalt kan der indlæses op til 126 tegn på en linie, men denne grænse kan ændres ved at tilskrive en ny værdi til den prædefinerede variabel buflen. Værdien skal være et heltal mellem 0 og 126. En sådan tilskrivning berører kun den næste indlæsning; når først denne er foretaget, bliver grænsen igen sat til 126. Et eksempel:

```
write('navn (op til 24 tegn)? ');
buflen:=24; read(navn);
```

16.2 readln proceduren

readln proceduren er identisk med read proceduren, bortset fra at efter endt læsning af den sidste variabel, bliver resten af linien oversprunget, dvs. alle tegn til og med den næste CR/LF sekvens bliver læst. Formatet af proceduresætningen er:

```
readln(v1,v2,...,vn)   eller   readln(f,v1,v2,...,vn)
```

Efter et kald til readln gælder der, at det næste kald til read eller readln starter med det første tegn på den næste linie. eoln er altid falsk efter

et kald til readln, med mindre eof er sand. readln kan også anvendes uden variable:

readln eller readln(f)

Når readln anvendes i sin korte form i {$B+} tilstanden, er den eneste forskel fra et tilsvarende kald til read, at en CR/LF sekvens udskrives til skærmen efter endt indlæsning af linien.

16.3 write proceduren

write proceduren anvendes til at udskrive strenge, booleans, og numeriske værdier. Formatet af proceduresætningen er:

write(p1,p2,...,pn) eller write(f,p1,p2,...,pn)

hvor p1,p2,...,pn angiver såkaldte write–parametre, og f angiver en textfil. Write–parametrene kan antage forskellige former, alt efter hvilke datatyper der udskrives (m, n og i angiver udtryk af typen integer, ch angiver et udtryk af typen char, s angiver et strengudtryk, b angiver et udtryk af typen boolean, og r angiver et udtryk af typen real):

ch Tegnet ch udskrives.

ch:n Tegnet ch udskrives højrejusteret i et felt på n tegn.

s Strengudtrykket s udskrives. Bemærk, at tegn–arrays også kan udskrives, da disse er kompatible med strenge.

s:n Strengudtrykket s udskrives højrejusteret i et felt på n tegn.

b TRUE eller FALSE udskrives alt efter b's værdi.

b:n TRUE eller FALSE udskrives højrejusteret i et felt på n tegn.

i Værdien af udtrykket i udskrives i frit format.

i:n Værdien af udtrykket i udskrives højrejusteret i et felt på n tegn.

r Værdien af udtrykket r udskrives i exponentiel notation. Feltbredden er 18 tegn i PolyPascal–80 og standardversionen af PolyPascal–86:

r>=0.0: bbd.ddddddddddEtdd
r<0.0: b-d.ddddddddddEtdd

Feltbredden er 25 tegn i Business versionen af PolyPascal-86:

r>=0.0: bbd.ddddddddddddddddEtddd
r<0.0: b-d.ddddddddddddddddEtddd

Feltbredden er 23 tegn i 8087 versionen af PolyPascal-86:

r>=0.0: bbd.ddddddddddddddEtddd
r<0.0: b-d.ddddddddddddddEtddd

b angiver et blanktegn, d angiver et ciffer og t angiver enten '+' eller '-'.

r:n Værdien af udtrykket r udskrives i exponentiel notation i et felt på n tegn. Det generelle format er:

r>=0.0: <blanks>d.<digits>Et<expo>
r<0.0: <blanks>-d.<digits>Et<expo>

hvor <blanks> er nul eller flere blanktegn, d er et ciffer, <digits> er mellem 1 og 10 cifre, t er '+' eller '-' og <expo> er to cifre. I Business versionen af PolyPascal-86 er <digits> mellem 1 og 17 cifre. I 8087 versionen af PolyPascal-86 er <digits> mellem 1 og 14 cifre og <expo> tre cifre.

r:n:m Værdien af udtrykket r udskrives højrejusteret i fastkommanotation i et felt på n tegn med m decimaler. m skal være mellem 0 og 24, ellers vælges exponentiel notation.

I Business versionen af PolyPascal-86 kan standardfunktionen fmt bruges sammen med write proceduren til at lave formaterede udskrifter. Et eksempel:

```
write(fmt('Total: ##.###.###,##',total));
```

fmt funktionen er beskrevet i afsnit 8.4.1.

16.4 writeln proceduren

writeln proceduren er identisk med write proceduren, bortset fra, at der udskrives en CR/LF sekvens efter den sidste værdi. Formatet af proceduresætningen er:

writeln(p1,p2,...,pn) eller writeln(f,p1,p2,...,pn)

Et linieskift kan udskrives alene ved at udelade write–parametrene:

writeln eller writeln(f)

17. Afbrydning af programmer

Et PolyPascal program kan afbrydes på to måder: Under ind- og udlæsning eller under kørsel.

17.1 Programafbrydning under ind- og udlæsning

Når PolyPascal modtager et Ctrl-S tegn fra tastaturet under udlæsning til skærmen, bliver udskriften midlertidigt stoppet, indtil der tastes endnu et tegn på tastaturet. Hvis PolyPascal modtager et Ctrl-C tegn under en ind- eller udlæsning, afbrydes programmet. Herved fremkommer meddelelsen:

```
USER INTERRUPT AT PC=aaaa
Program terminated
```

hvorefter der returneres til PolyPascal ('>>' klartegnet) eller til CP/M eller MS-DOS. I lighed med kørselsfejl og I/O fejl kan det sted programmet blev afbrudt findes med FIND kommandoen.

I visse tilfælde er det ikke ønskeligt, at brugeren kan afbryde programmet som beskrevet ovenfor. Denne facilitet kan derfor slås fra ved hjælp af C compilerdirektivet. Ved start af compileren vælges {$C+}, og i denne stilling kan programmet afbrydes under ind- og udlæsninger. Hvis et {$C-} direktiv placeres i begyndelsen af programmet (før erklæringsdelen), bliver Ctrl-S og Ctrl-C behandlet som alle andre tegn.

17.2 Programafbrydning under kørsel

Hvis et fejlbehæftet program går i en uendelig løkke, vil den eneste måde hvorpå det kan afbrydes normalt være et tryk på systemets RESET-knap, hvilket i værste tilfælde kan betyde, at kildeteksten går tabt. I PolyPascal er det imidlertid muligt at instruere compileren om, at den programkode der genereres lejlighedsvis skal checke tastaturet for at se, om brugeren ønsker at afbryde programmet. Dette gøres ved hjælp af U compilerdirektivet. Ved start af compileren vælges {$U-}, og i denne stilling vil det færdige program ikke kunne afbrydes. I den modsatte stilling, {$U+}, genererer compileren et kald til et check-rutine før hver programsætning. I modsætning til C direktivet kan U

direktivet frit anvendes gennem kildeteksten, og alle sætninger der oversættes i {$U+} stillingen vil kunne afbrydes under kørslen ved at taste Ctrl–C.

Bemærk, at sætninger, der er oversat i {$U+} stillingen, udføres mærkbart langsommere end sætninger, der ikke kan afbrydes.

PolyPascal–80 anvender en RST 38H instruktion til kald af Ctrl–C check–rutinen og PolyPascal–86 anvender en INT 3 instruktion. Disse instruktioner er normalt ikke reserverede, da CP/M og MS–DOS debuggerne (DDT, DDT86 og DEBUG) bruger dem til breakpoints.

18. Include–filer

Det kan ske, at et programs kildetekst bliver så stor, at den ikke kan være i PolyPascal editorens buffer på en gang. I så tilfælde kan kildeteksten inddeles i mindre segmenter, der sammensættes under oversættelsen af programmet ved hjælp af "include–fil" compilerdirektiver.

Syntaksen for et compilerdirektiv, der instruerer compileren om, at medtage en anden kildetekst i oversættelsen af den nuværende kildetekst, er som følger:

```
{$I filnavn}
```

hvor filnavn er et CP/M eller MS–DOS filnavn. Hvis filtypen udelades, bliver '.PAS' automatisk valgt. Blanktegn foran og efter filnavnet ignoreres. Det anbefales, at lade include–file compilerdirektivet stå alene på linien.

Compileren tillader ikke nestede include–filer. Der rapporteres en fejl, hvis en include–fil forsøger at inkludere en anden fil.

Include–filer kan også anvendes til at opbygge biblioteker af "macrodefinitioner" på disken. Sådanne macrodefinitionerer kan medtages i oversættelsen af et program når som helst det er påkrævet, blot ved at angive deres filnavn i et "include–fil" compilerdirektiv. Antag, som et eksempel, at den følgende macrodefinition findes på disken under filnavnet A:MINMAX.LIB:

```
TYPE tal = 0..99;

FUNCTION max(a,b: tal): tal;
BEGIN if a>b THEN max:=a ELSE max:=b END;

FUNCTION min(a,b: tal): tal;
BEGIN if a<b THEN max:=a ELSE max:=b END;
```

Et program, der anvender typen tal og funktionerne max og min, kan da skrives som:

```
PROGRAM minmax;
{$I A:MINMAX.LIB}
VAR
  x,y: tal;
BEGIN
  write('skriv to tal: '); readln(x,y);
  writeln('det største tal er ',max(x,y));
  writeln('det mindste tal er ',min(x,y));
END.
```

Da de enkelte sektioner i en erklæringsdel kan skrives i vilkårlig rækkefølge og eventuelt flere gange, kan macrodefinitioner både erklære konstanter, typer, variable og underprogrammer.

PolyPascal bevarer tilstanden af alle compilerflag og registre under læsning af en include–fil. En include–fil må naturligvis godt indeholde compilerdirektiver, men disse direktiver vil da kun gælde lokalt.

19. Kædning af programmer

I PolyPascal er det muligt for et program at starte et andet. Dette kan udnyttes når programmer bliver så store, at de ikke kan være i lageret på en gang. Kædning af programmer udføres via execute og chain procedurerne. Formatet af kald til disse procedurer er:

execute(f) og chain(f)

hvor f er en filvariabel (af enhver filtype), der tidligere er tilskrevet et filnavn ved hjælp af assign proceduren. Forudsat at det givne program eksisterer, bliver det læst ind i lageret og startet.

execute proceduren bruges til at indlæse og opstarte en anden PolyPascal programfil ('.COM' eller '.CMD' fil). chain proceduren kan kun anvendes til at aktivere PolyPascal objektfiler ('.OBJ' filer), dvs. filer, der er genereret med PolyPascal's OBJECT kommando.

Yderligere detaljer om PROGRAM og OBJECT kommandoerne gives i PolyPascal brugermanualen. Kort fortalt er forskellen på disse kommandoer, at PROGRAM kommandoen medtager både run–time programdelen og programmets kode i filen (og skaber dermed en programfil, der kan køre helt af sig selv), medens OBJECT kommandoen kun medtager programkoden. En fil, der er genereret af OBJECT kommandoen, kan derfor kun køre, hvis run–time programdelen allerede findes i lageret.

Hvis den fil, der refereres til i et kald til execute eller chain, ikke findes, gives en I/O fejl, med mindre kaldet er oversat i {$I–} stillingen. I sidstnævnte tilfælde fortsætter programmet med sætningen efter kaldet, og iores funktionen giver nu en værdi forskellig fra nul. Denne værdi skal undersøges af programmet, før yderligere I/O operationer kan foretages. Et eksempel:

```
VAR
  f: FILE;
  i: integer;
BEGIN
  assign(f,'B:PROG3.COM'); {$I-} execute(f) {$I+};
  i:=iores;
  writeln('B:PROG3.COM findes ikke.');
END.
```

Brugere af PolyPascal–86 skal erstatte '.COM' med '.CMD' ovenfor.

Data kan overføres mellem kædede programmer ved hjælp af fælles globale variable eller absolut adresserede variable. Naturligvis kan en fil også anvendes, men denne metode er noget langsommere end de to andre.

Dataoverførsel ved hjælp af fælles globale variable kræver, at erklæringerne af de fælles variable foretages som de første erklæringer i begge programmer, og desuden, at begge programmer er oversat til den samme lagerstørrelse. Et eksempel:

```
PROGRAM hovedprog;
VAR
  i,j,k: integer;
  f: FILE;
BEGIN
  readln(i,j); k:=i*j;
  assign(f,'A:UNDERPRG.COM'); execute(f);
END.

PROGRAM underprg;
VAR
  i,j,k: integer;
BEGIN
  writeln(i,' gange ',j,' er ',k);
END.
```

Brugere af PolyPascal–86 skal erstatte '.COM' med '.CMD' ovenfor.

Dataoverførsel ved hjælp af absolut adresserede variable foregår typisk ved, at man definerer en postvariabel, der indeholder felter med alle de informationer, der skal overføres, og derefter erklærer en variabel af denne type på en fast adresse (ved hjælp af AT specifikationen).

Bemærk, at det ikke kan lade sig gøre at kæde programmer i direct mode, eller, med andre ord, at et program, der er startet fra PolyPascal med RUN kommandoen, ikke må starte andre programmer.

Hvis et program skal kunne afgøre om det blev startet fra operativsystemet eller fra et kald til execute, kan det eventuelt erklære et flag på offset $80 i kodesegmentet (MS–DOS) eller datasegmentet (CP/M–86), og undersøge dets værdi ved opstart. Hvis programmet blev startet fra operativsystemet, er flagets værdi altid mellem 0 og 127, da det også anvendes til at gemme længden af kommandolinien. Hvis programmet startes fra et andet program, bør dette program ligeledes erklære det omtalte flag samt gemme en værdi over 127 i det. Flaget kan da anvendes i en test som vist nedenfor:

```
PROGRAM prog1;
VAR
  f: FILE;
  startflag: byte AT cseg:$80;
BEGIN
  startflag:=255;
  assign(f,'A:PROGRAM2.COM'); execute(f);
END;

PROGRAM prog2;
VAR
  startflag: byte AT cseg:$80;
BEGIN
  IF startflag=255 THEN
  writeln('startet fra execute eller chain.') ELSE
  writeln('startet fra operativsystem.');
END.
```

Dette eksempel er skrevet til MS–DOS versionen af PolyPascal–86. Brugere af PolyPascal–86 under CP/M skal erstatte 'cseg' identifieren ovenfor med 'dseg', og erstatte '.COM' med '.CMD'. Brugere af PolyPascal–80 skal fjerne 'cseg:' delen fra AT specifikationen.

Nedenfor følger bemærkninger til kædning af programmer i de enkelte versioner af PolyPascal.

PolyPascal–86
I PolyPascal–86 forbliver størrelsen og placeringen af et programs segmenter (kode, data og stak) konstant ved et kald til execute eller chain.

execute og chain udskifter blot koden i kodesegmentet, mens data og staksegmenterne forbliver uændrede. execute kan derfor ikke anvendes til at starte "fremmede" programmer, dvs. programfiler, der ikke er genereret af PolyPascal compileren. Det er op til programmøren at sikre, at "rod–programmet" (dvs. det program, der blev startet fra operativsystemet), allokerer kode, data og staksegmenter der er store nok til det største program, der senere hen skal aktiveres. Størrelsen af et programs segmenter kan angives ved hjælp af parametre i en PROGRAM kommando.

PolyPascal–80
execute proceduren kan bruges til at indlæse og opstarte enhver CP/M programfil ('.COM' fil). Filen læses ind i lageret startende fra adresse $100, og startes i adresse $100, som CP/M standarden foreskriver.

chain proceduren kan kun anvendes til at aktivere PolyPascal objektfiler ('.OBJ' filer), der er genereret med OBJECT kommandoen. Filen læses ind i lageret på startadressen for det nuværende program (dvs. på den adresse, der blev specificeret som startadresse, da det nuværende program blev oversat). Kravet for, at chain fungerer korrekt, er altså, at det nuværende program og det program, der aktiveres, er oversat til den samme startadresse.

20. In–line maskinkode

Til tider er det ønskeligt at skrive subrutiner eller dele af programmer i maskinkode for at optimere kørselshastigheden. Dette er muligt i PolyPascal ved hjælp af CODE sætninger.

En CODE sætning består af det reserverede ord CODE efterfulgt af et eller flere kodeelementer adskilt af kommaer. Et kodeelement består af et eller flere dataelementer adskilt af enten plus (+) eller minus (–) tegn. Et dataelement kan være en heltalskonstant, en variabelidentifier, en procedureidentifier, en funktionsidentifier eller en programtællerreference (skrevet som '*'). Et eksempel:

```
CODE 10,$2345,count+1,sort-*+2;
```

Hver kodeelement genererer en byte eller et ord (to bytes) kode. Værdien af et kodeelement bliver beregnet ved at addere henholdsvis subtrahere de enkelte dataelementer, det består af. Værdien af en variabel, en procedure eller en funktion, er adressen af denne. Værdien af programtælleren er adressen, hvortil næste byte vil blive genereret.

Et kodeelement genererer kun en enkelt byte, hvis det er i 8–bit området (0..255). Hvis værdien er udenfor dette område, eller hvis kodeelementet refererer til en adresse eller til programtælleren, genereres et ord (to bytes med mindst betydende byte først).

Tegnene '<' og '>' bruges til at tilsidesætte det ovenfor nævnte automatiske valg af en eller to bytes kode. Hvis et kodeelement starter med '<', genereres der kun en byte (den mindst betydende byte af kodeelementets værdi). Hvis et kodeelement starter med '>', genereres der altid et ord. Et eksempel:

```
CODE <$1234,>$44;
```

Dette eksempel genererer tre bytes: $34, $44, $00.

CODE sætninger kan frit blandes med andre sætninger i en bloks sætningsdel.

Eksemplerne der følger afhænger af hvilken version af PolyPascal, der anvendes.

PolyPascal-86
Når en variabelidentifier angives i en CODE sætning, svarer dens værdi til dens adresse. Globale variable lagres altid i datasegmentet, hvis segmentadresse findes i DS registeret. Lokale variable (dvs. variable, der er erklæret i det nuværende underprogram) lagres altid i staksegmentet, hvis segmentadresse findes i SS registeret. En lokal variabels offsetadresse er desuden altid relativ til BP (base page) registeret. Bemærk, at når BP registeret anvendes i et adresseudtryk, vælger 8086 processoren automatisk staksegmentet. Typeangivne konstanter lagres altid i kodesegmentet, hvis segmentadresse findes i CS registeret. CODE sætninger bør ikke forsøge at adressere variable, der hverken er erklæret i hovedprogrammet eller i det nuværende underprogram.

I det nedenfor viste program anvendes en CODE sætning til at definere en procedure, der konverterer alle bogstaver i en streng til store bogstaver.

```
PROGRAM teststore;
TYPE
str = STRING[64];
VAR
s: str;

PROCEDURE store(VAR streng: str);
BEGIN CODE
  $C4,$BE,strg,       {     LES  DI,strg[BP]                  }
  $26,$8A,$0D,        {     MOV CL,ES:[DI]                    }
  $FE,$C1,            {     INC  CL                           }
  $FE,$C9,            { L1: DEC  CL                           }
  $74,$13,            {     JZ   L2                           }
  $47,                {     INC  DI                           }
  $26,$80,$3D,$61,    {     CMP ES:BYTE PTR [DI],'a'          }
  $72,$F5,            {     JB   L1                           }
  $26,$80,$3D,$7A,    {     CMP ES:BYTE PTR [DI],'z'          }
  $77,$EF,            {     JA   L1                           }
  $26,$80,$2D,$20,    {     SUB  ES:BYTE PTR [DI],20H         }
  $EB,$E9;            {     JMP  SHORT L1                     }
END;                  { L2:                                   }

BEGIN
  write('indtast en streng: '); readln(s);
  store(s); writeln(s);
END.
```

CODE sætninger skal bevare BP, SP, CS, DS og SS registrene.

PolyPascal-80
I det nedenfor viste program anvendes en CODE sætning til at definere en procedure, der konverterer alle bogstaver i en streng til store bogstaver.

```
PROGRAM teststore; {$A+}
TYPE
  str = STRING[64];
VAR
  s: str;

PROCEDURE store(VAR streng: str);
BEGIN CODE
  $2A,streng,          {     LD   HL,(streng)          }
  $46,                 {     LD   B,(HL)               }
  $04,                 {     INC  B                    }
  $05,                 { L1: DEC  B                    }
  $CA,*+20,            {     JP   Z,L2                 }
  $23,                 {     INC  HL                   }
  $7E,                 {     LD   A,(HL)               }
  $FE,$61,             {     CP   'a'                  }
  $DA,*-9,             {     JP   C,L1                 }
  $FE,$7B,             {     CP   'z'+1                }
  $D2,*-14,            {     JP   NC,L1                }
  $D6,$20,             {     SUB  20H                  }
  $77,                 {     LD   (HL),A               }
  $C3,*-20;            {     JP   L1                   }
END;                   { L2: EQU  $                    }

BEGIN
  write('indtast en streng: '); readln(s);
  store(s); writeln(s);
END.
```

Bemærk, at den eneste grund til, at der anvendes JP instruktioner, er, at det da bliver muligt at demonstrere programtællerreferencer. I den ovenstående rutine ville det naturligvis være kortere at anvende JR instruktioner.

CODE sætninger må anvende alle CPU registre (bemærk dog, at stack-pointeren (SP) skal have det samme indhold ved indgang som ved udgang).

21. Systemkald

21.1 PolyPascal–86 systemkald

I PolyPascal–86 foretages alle systemkald via en standard procedure ved navn swint (software interrupt). swint proceduren tager to argumenter. Det første argument skal være en heltalskonstant mellem 0 og 255, og angiver nummeret på det interrupt, der skal udføres. Det andet argument skal være en variabel af typen regpack. Definitionen af regpack er:

```
TYPE
regpack = RECORD
            ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
          END;
```

Før swint udfører selve interrupt instruktionen, læses registrene AX, BX, CX, DX, BP, SI, DI, DS og ES fra regpack variablen (bemærk at flagene ikke initialiseres). Når interrupt instruktionen returnerer, gemmes registrenes indhold i variablen. Det nedenfor viste program bruger swint proceduren til at udføre et MS–DOS systemkald. Programmet overfører værdien 9 (print string) i AH registeret, og adressen på en streng i DS:DX registerparret.

```
PROGRAM test_systemkald;
TYPE
  str18 = ARRAY[1..18] OF char;
CONST
  streng: str18 = 'Systemkald virker$';
VAR
  regs: RECORD
          ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
        END;
BEGIN
  regs.ax:=$0900;
  regs.dx:=ofs(streng);
  regs.ds:=seg(streng);
  swint($21,regs);
END.
```

Informationer om MS-DOS systemkald findes i manualen "MS-DOS Operating System Programmer's Reference Manual".

Hvis det ovenfor viste program skal anvendes under CP/M, skal der i stedet anvendes en INT $E0 instruktion og funktionskoden skal overføres i CL registeret:

```
BEGIN
  regs.cx:=9;
  regs.dx:=ofs(streng);
  regs.ds:=seg(streng);
  swint($E0,regs);
END.
```

Informationer om CP/M-86 systemkald (BDOS kald) findes i manualen "CP/M-86 Operating System System Guide".

21.2 PolyPascal-80 systemkald

Til kald af CP/M's BDOS og BIOS rutiner findes der i PolyPascal to standardprocedurer, kaldet bdos og bios, og fire standardfunktioner, kaldet bdos, bios, bdosb, og biosb. Disse rutiner bør kun anvendes af øvede programmører, der fuldt ud forstår deres implikationer.

bdos(f,p) Denne procedure bruges til at kalde CP/M BDOS rutiner. f og p er udtryk af typen integer (hvis rutinen ikke kræver en indgangsparameter, kan p og det foranstående komma kan udelades). f loades ind i C registeret, p (hvis angivet) loades ind i DE registerparret, og derefter udføres et kald til adresse 5, hvilket aktiverer BDOS rutinen. bdos kan også anvendes som en funktion. I så tilfælde er resultatet (af typen integer) den værdi, der returneres i HL registerparret.

bios(f,p) Denne procedure bruges til at kalde BIOS rutiner. f og p er udtryk af typen integer (hvis rutinen ikke kræver en indgangsparameter, kan p og det foranstående komma udelades). f angiver nummeret på den rutine der skal kaldes, hvor 0 svarer til WBOOT, 1 til CONST, osv. (med andre ord, adressen på rutinen, der skal kaldes, udregnes ved at lægge f*3 til den adresse, der er inde-

holdt i adresse 1 og 2). Hvis p angives, loades denne værdi ind i BC registerparret før kaldet. bios kan også anvendes som en funktion. I så tilfælde er resultatet (af typen integer) den værdi, der returneres i HL registerparret.

bdosb(f,p) Denne funktion svarer til bdos, bortset fra, at resultatet er den værdi, der returneres i A registeret.

biosb(f,p) Denne funktion svarer til bios, bortset fra, at resultatet er den værdi, der returneres i A registeret.

Informationer om BDOS og BIOS rutiner findes i manualerne "CP/M Interface Guide" og "CP/M Alteration Guide", der begge udgives af Digital Research.

22. Brugerdefinerede I/O drivere

Til visse formål er det ønskeligt eller ligefrem nødvendigt for et program at definere dets egne low–level I/O drivere, dvs. rutiner der foretager den grundlæggende ind– og udlæsning af tegn fra og til ydre enheder. I PolyPascal findes de følgende grundlæggende I/O drivere (bemærk, at disse subrutiner ikke er tilgængelige som standardprocedurer og funktioner):

```
FUNCTION const: boolean;
FUNCTION conin: char;
PROCEDURE conout(ch: char);
PROCEDURE lstout(ch: char);
PROCEDURE auxout(ch: char);
FUNCTION auxin: char;
PROCEDURE usrout(ch: char);
FUNCTION usrin: char;
```

const rutinen kaldes af keypress funktionen, conin og conout rutinerne bruges af CON: og KBD: enhederne, lstout rutinen bruges af LST: enheden, auxout og auxin rutinerne bruges af AUX: enheden, og usrout og usrin rutinerne bruges af USR: enheden.

I MS–DOS versionen bruger const, conin og conout normalt systemkald nummer 6 (eller 11, 8 og 2 i {$C–} tilstanden), lstout nummer 5, auxout nummer 4, auxin nummer 3, og usrout og usrin nummer 6 (eller 2 og 8 i {$C–} tilstanden).

CP/M versionerne af PolyPascal anvender CP/M systemets BIOS (dvs. const bruger CONST, conin bruger CONIN, conout bruger CONOUT, lstout bruger LIST, auxout bruger PUNCH, auxin bruger READER, usrout bruger CONOUT, og usrin bruger CONIN).

Disse definitioner kan imidlertid ændres af et program ved at tilskrive adressen (PolyPascal–80) eller offsetadressen (PolyPascal–86) på en driver til en af de følgende standardvariable ('addr' endelsen bruges i PolyPascal–80 og 'ofs' endelsen i PolyPascal–86):

csaddr	csofs	adressen på const funktionen
ciaddr	ciofs	adressen på conin funktionen
coaddr	coofs	adressen på conout proceduren
loaddr	loofs	adressen på lstout proceduren
aoaddr	aoofs	adressen på auxout proceduren
aiaddr	aiofs	adressen på auxin funktionen
uoaddr	uoofs	adressen på usrout proceduren
uiaddr	uiofs	adressen på usrin funktionen

I PolyPascal–86 skal rutinerne altid befinde sig i kodesegmentet, der adresseres via CS registeret.

Driverprocedurer og driverfunktioner skal overholde de ovennævnte definitioner, dvs. en const driver skal være en boolean funktion, en conin, auxin eller usrin driver skal være en char funktion, og en conout, lstout, auxout eller usrout driver skal være en procedure med en value–parameter af typen char.

Det nedenfor viste program definerer og aktiverer en ny driver for LST: enheden. Udover at udskrive tegn holder den nye driver automatisk styr på printerens linie og position. Foran hver linie udskrives en venstremargin, der består af et brugerdefineret antal blanktegn, og perforeringen mellem siderne overspringes automatisk. Desuden bliver form–feed tegn omsat til et passende antal line–feed tegn. Enkelttegn udskrives fra driveren ved at kalde MS–DOS funktion 5 (List Output).

```
PROGRAM listdriver;
CONST
  sidelaengde = 72;   {sidelængde i linier}
  bundmargin = 6;     {overspring perforering}
  venstremargin = 8;  {venstremargin}
VAR
  lstlin,lstpos: integer;

PROCEDURE prchr(ch: char);
VAR
  regs: RECORD
          ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
        END;
BEGIN
  regs.ax:=$0500; regs.dx:=ord(ch); swint($21,regs);
END;
```

```
PROCEDURE lstout(ch: char);
VAR i: integer;
BEGIN
  IF ch>=' ' THEN
  BEGIN
    IF lstpos=0 THEN
    BEGIN
      FOR i:=1 TO venstremargin DO prchr(' ');
      lstpos:=venstremargin;
    END;
    prchr(ch); lstpos:=lstpos+1;
  END ELSE
  IF ch=@13 THEN
  BEGIN
    prchr(@13); lstpos:=0;
  END ELSE
  IF ch=@10 THEN
  BEGIN
    prchr(@10); lstlin:=lstlin+1;
    IF lstlin=sidelaengde-bundmargin THEN
    BEGIN
      FOR i:=1 TO bundmargin DO prchr(@10);
      lstlin:=0;
    END;
  END ELSE
  IF ch=@12 THEN
  BEGIN
    FOR i:=lstlin TO sidelaengde-1 DO prchr(@10);
    lstlin:=0;
  END;
END;

BEGIN
  loofs:=ofs(lstout); lstpos:=0; lstlin:=0;
  writeln(lst,'LST DRIVER TEST:');
  writeln(lst,'Dette giver tre blanke linier...');
  write(lst,@10@10@10);
  writeln(lst,'Dette giver et sideskift...');
  write(lst,@12);
END.
```

Det ovenfor viste program er skrevet til MS-DOS versionen af Poly-

Pascal–86. I CP/M–86 versionen af PolyPascal–86 skal prchr proceduren se således ud:

```
PROCEDURE prchr(ch: char);
  VAR
  regs: RECORD
          ax,bx,cx,dx,bp,si,di,ds,es,flags: integer;
        END;
BEGIN
  regs.cx:=5; regs.dx:=ord(ch); swint($E0,regs);
END;
```

I PolyPascal–80 skal prchr se således ud:

```
PROCEDURE prchr(ch: char);
BEGIN
  bios(4,ord(ch));
END;
```

I PolyPascal–80 skal der desuden indsættes et {$A+} compilerdirektiv i begyndelsen af programmet, og den første linie i hovedprogrammet ændres til:

```
loaddr:=addr(lstout); lstpos:=0; lstlin:=0;
```

I begyndelsen af programmet sættes printerens linie og position til nul, og LST: driver adressen ændres til adressen på den brugerdefinerede outputrutine. Herefter udskrives en række strenge til lst filen (der er prædefineret, og altid refererer til LST: enheden). Disse strenge bliver af systemet overgivet til lstout proceduren et tegn ad gangen. Bemærk, at brugerdefinerede I/O drivere under ingen omstændigheder må kalde read, readln, write og writeln procedurerne. Bemærk desuden, at hvis et program, der indeholder brugerdefinerede drivere, aktiverer andre programmer via kald til execute eller chain, så skal disse programmer også definere og aktivere driverne.

Brugerdefinerede drivere skal altid oversættes med U compilerdirektivet fra, dvs. i {$U–} tilstanden. I PolyPascal–80 bør A compilerdirektivet desuden være til, dvs. {$A+}.

23. Interne dataformater

I de følgende beskrivelser angiver symbolet '@' den første byte, der optages af den givne variabel. De metoder der bruges til at reservere lagerplads til variable afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86
Hovedprogrammets variable, dvs. de variable, der er erklæret i hovedprogrammets erklæringsdel, lagres altid i datasegmentet, hvis segmentadresse findes i DS registeret. Typeangivne konstanter lagres i kodesegmentet, hvis segmentadresse findes i CS registeret. Procedurers og funktioners variable lagres i staksegmentet, hvis segmentadresse findes i SS registeret. I modsætning til PolyPascal–80, bliver lokale variable i PolyPascal–86 først allokeret når et underprogram kaldes, og når det returnerer, forsvinder variablene igen. Lokale variable er altså dynamiske, dvs. deres adresse kendes ikke under oversættelsen af programmet.

Variable er altid fuldstændig indeholdt i deres segment, dvs. offsetadressen på den sidste byte, der optages af en variabel, er aldrig større end $FFFF. Af denne grund kan en enkelt variabel aldrig optage mere end 64K bytes.

I PolyPascal–86 returnerer standardfunktionen addr en pointer til en given variabel, og standardfunktionerne ofs og seg returnerer en given variabels offsetadresse og segmentadresse.

PolyPascal–80
PolyPascal–80 compileren reserverer plads til variable fra toppen af lageret og nedefter. Alle variable, både hovedprogrammets og underprogrammernes, er statisk allokerede, dvs. de findes på de samme adresser hele tiden.

I PolyPascal–80 kan standardfunktionen addr bruges til at finde adressen på en given variabel.

23.1 Grundlæggende datatyper

Variable af de grundlæggende datatyper kan grupperes i strukturer, men dette berører ikke deres interne formater.

23.1.1 Skalarer

Integer delintervaller, hvor begge grænser er mellem 0 og 255, booleans, tegn (char variable), og selvdefinerede skalarer, med mindre end 256 elementer, optager en byte i lageret, der angiver den ordinale værdi af variablen.

Integers, integer delintervaller, hvor en eller begge grænser er udenfor området 0..255, og selvdefinerede skalarer, med mere end 256 mulige værdier, optager to bytes i lageret. Disse bytes udgør en 16–bit 2's komplement værdi. Den mindst betydende byte gemmes først.

23.1.2 Reals

En variabel af typen real optager seks bytes, hvoraf fem bytes (40 bit) udgør mantissen og en byte (8 bits) udgør exponenten. Exponenten gemmes i den første byte, og mantissen i de følgende fem bytes, med den mindst betydende byte først:

```
@+0   Exponent.
@+1   Mantissens mindst betydende byte.
:
@+5   Mantissens mest betydende byte.
```

Exponenten er et binært tal med et offset på $80, der angiver den potens af 2, mantissen skal multipliceres med. Således svarer en exponent på $84 til, at mantissen skal multipliceres med 2^($84–$80) = 2^4 = 16. Hvis exponenten er $00, opfattes hele tallet som værende 0. Værdien af mantissen findes ved, at dividere det fortegnsløse 40–bits heltal, der udgøres af de fem bytes, med 2^40. Mantissen er altid normaliseret, dvs. den mest betydende bit (bit 7 i @+5) skal altid opfattes som værende 1. Fortegnet gemmes i den mest betydende bit: Er denne 1, er tallet negativt, er den 0, er tallet positivt.

Business versionen af PolyPascal–86 bruger 10 bytes til at lagre en real, hvilket giver en flydende værdi med 18 cifres mantisse i BCD (binary coded decimal) format, og en 7–bits titalsexponent. Tallets fortegn og exponent er gemt i den første byte, og mantissen i de næste 9 bytes, med den mindst betydende byte først.

```
@+0   Exponent og fortegn.
@+1   Mindst betydende byte af mantissen.
:
@+9   Mest betydende byte af mantissen.
```

Den mest betydende bit i den første byte indeholder fortegnet for tallet. 0 betyder positivt og 1 betyder negativt. De resterende 7 bits indeholder exponenten på binær form med et offset på $3F. En exponent på $41 angiver derfor at mantissen skal ganges med 10^($41-$3F) = 10^2 = 100. Hvis den første byte er 0, antages hele værdien at være 0. Hver byte i mantissen (startende med den tiende=@+9) indeholder to cifre i BCD format, med det mest betydende ciffer i de fire mest betydende bits. Det første ciffer angiver tiendedele, det andet hundrededele, det tredje ciffer (i byte @+8) tusindedele, osv. Mantissen er altid normaliseret, således at det første ciffer aldrig er 0.

8087 versionen af PolyPascal-86 bruges 8 bytes til at lagre en real. Det anvendte format er 8087'erens "long real" format. I bogen "iAPX 86,88 Users Manual", der udgives af Intel Corporation, findes en komplet beskrivelse af 8087 co-processoren.

23.1.3 Strenge

En streng optager dens maksimale længde plus en byte i lageret. Den første byte angiver længden, og de efterfølgende bytes indeholder tegnene, med det første tegn på den laveste adresse. I den nedenfor viste figur angiver n den nuværende længde af strengen og m den maksimale længde.

```
@+0     Nuværende længde (n).
@+1     Første tegn.
@+2     Andet tegn.
:
@+n     Sidste tegn.
@+n+1   Ubrugt.
:
@+m     Ubrugt.
```

23.1.4 Mængder

Et element i en mængde optager en bit, og da der aldrig er mere end 256 elementer i en mængde, fylder en mængde aldrig mere en 32 (256/8) bytes.

Hvis en mængde indeholder mindre end 256 elementer, er det givet, at nogle af de 256 bits permanent vil være nul, og det er derfor ikke nødvendigt at gemme disse bits. Udfra et lager-økonomisk synspunkt ville den mest effektive måde at gemme mængder på da være at "bort-

skære" alle unødige bits og rotere de resterende bits, således at det første element i mængden optager den første bit i den første byte. Imidlertid er sådanne rotationer relativt tidskrævende, og PolyPascal implementerer derfor et kompromis: Kun de bytes, der er statisk nul (dvs. de bytes hvoraf ingen bits bruges til at repræsentere mængden), gemmes ikke. Denne kompressionsmetode er meget hurtig og i de fleste tilfælde lige så effektiv som den førstnævnte.

Antallet af bytes, der optages af en given mængdevariabel, kan beregnes af (max DIV 8)–(min DIV 8)+1, hvor min og max er de nedre og øvre grænser i den grundlæggende skalare type. Lageradressen på et givet element beregnes af:

memaddr = addr+(e DIV 8)–(min DIV 8)

og bitadressen, i byten på lageradressen memaddr, beregnes af:

bitaddr = e MOD 8

hvor e er elementets ordinale værdi.

23.1.5 Fil interface blokke

Til enhver filvariabel i et program svarer der en fil interface blok (FIB) i datalageret. Formatet af en FIB afhænger af hvilken version af PolyPascal der anvendes.

PolyPascal–86 til MS–DOS
Formatet af en FIB i MS–DOS versionen af PolyPascal–86 er som følger:

@+0	Filnummer (LSB).
@+1	Filnummer (MSB).
@+2	Recordlængde (LSB) eller flagbyte.
@+3	Recordlængde (MSB) eller tegnbuffer.
@+4	Bufferadresse (LSB).
@+5	Bufferadresse (MSB).
@+6	Bufferstørrelse (LSB).
@+7	Bufferstørrelse (MSB).
@+8	Bufferpointer (LSB).
@+9	Bufferpointer (MSB).
@+10	Buffer slutadresse (LSB).
@+11	Buffer slutadresse (MSB).

@+12 Første byte i filnavn (path).
:
@+75 Sidste byte i filnavn (path).

Ordet i @+0 inderholder det 16-bit filnummer (handle), MS-DOS returnerede, da filen blev åbnet (eller $FFFF når filen er lukket). For typeangivne og typeløse filer, indeholder ordet i @+2 recordlængden i bytes (0 hvis filen er lukket), medens de 4 ord fra @+4 til @+11 er ubrugte.

For textfiler er indeholdet af flagbyten i @+2 følgende:

Bit 0..3 Filtype.
Bit 5 Forudlæst-tegn flag
Bit 6 Skrivning tilladt flag.
Bit 7 Læsning tilladt flag.

Filtype 0 angiver en diskfil, medens 1 til 5 angiver PolyPascals logiske I/O enheder (CON:, KBD:, LST:, AUX:, og USR:). Bit 5 er sat hvis tegnbufferen i @+3 indeholder et læst men ubrugt tegn, bit 6 er sat hvis skrivning er tilladt, og bit 7 er sat hvis læsning er tilladt.

De fire ord fra @+4 til @+11 angiver filbufferens adresse, dens størrelse, adressen på det næste tegn der skal læses eller skrives, samt adressen på den første byte efter bufferen. Bufferen er altid i samme segment som FIB'en, og starter sædvanligvis i @+76. Når en textfil arbejder på en logisk enhed, er det kun de fire første bytes i FIB'en der anvendes.

Filnavnet (path'en) er en ASCIZ streng (en streng afsluttet med en chr(0) byte) på op til 63 tegn.

PolyPascal–86 til CP/M–86
Formatet af en FIB i CP/M versionen af PolyPascal–86 er som følger:

@+0	Flagbyte.
@+1	Tegnbuffer.
@+2	Antal records i filen eller bufferadresse (LSB).
@+3	Antal records i filen eller bufferadresse (MSB).
@+4	Recordlængde eller bufferstørrelse (LSB).
@+5	Recordlængde eller bufferstørrelse (MSB).
@+6	Bufferpointer (LSB).
@+7	Bufferpointer (MSB).
@+8	Recordnummer eller buffer slutadresse (LSB).
@+9	Recordnummer eller buffer slutadresse (MSB).
@+10	Ubrugt (reserveret).
@+11	Ubrugt (reserveret).
@+12	Første byte i CP/M FCB'en.
:	
@+47	Sidste byte i CP/M FCB'en.
@+48	Første byte i filbufferen.
:	
@+175	Sidste byte i filbufferen.

Formatet af flagbyten i @+0 er følgende:

Bit 0..3	Filtype.
Bit 4	Buffer ikke læst fra disk flag.
Bit 5	Buffer ikke skrevet til disk flag eller forudlæst–tegn flag.
Bit 6	Skrivning tilladt flag.
Bit 7	Læsning tilladt flag.

Filtype 0 angiver en diskfil, medens 1 til 5 angiver PolyPascals logiske I/O enheder (CON:, KBD:, LST:, AUX:, og USR:). For typeangivne filer er bit 4 sat hvis bufferens indhold er udefineret, og bit 5 sat hvis data er blevet skrevet i bufferen, men ikke i på disken. For textfiler er bit 5 er sat hvis tegnbufferen i @+3 indeholder et læst men ubrugt tegn. Bit 6 er sat hvis skrivning er tilladt, og bit 7 er sat hvis læsning er tilladt.

For typeangivne og typeløse filer gælder, at de fire ord fra @+2 til @+9 angiver antallet af records i filen, recordlængden i bytes, bufferpointeren, og det nuværende recordnummer. Bufferpointeren i en typeangiven fils FIB indeholder et index (0..127) til bufferen i @+48. En typeløs fils FIB har ingen buffer, og bufferpointeren bruges derfor ikke. Længden af en typeløs fils FIB er kun 48 bytes.

For textfiler gælder, at de fire ord fra @+2 til @+9 angiver filbufferens adresse, dens størrelse, adressen på det næste tegn der skal læses eller skrives, samt adressen på den første byte efter bufferen. Bufferen er altid i samme segment som FIB'en, og starter sædvanligvis i @+48. Bufferen kan eventuelt være større end angivet, men den er mindst 128 bytes, og altid et multiplum af 128. Når en textfil arbejder på en logisk enhed, er det kun de to første bytes i FIB'en der anvendes.

PolyPascal–80 for CP/M–80
Formatet af en FIB i PolyPascal–80 versionen er som følger:

@+0	Flag–byte.
@+1	Tegnbuffer.
@+2	Bufferpointer (LSB).
@+3	Bufferpointer (MSB).
@+4	Antal records i filen (LSB).
@+5	Antal records i filen (MSB).
@+6	Recordlængde (LSB).
@+7	Recordlængde (MSB).
@+8	Nuværende recordnummer (LSB).
@+9	Nuværende recordnummer (MSB).
@+10	Ubrugt (reserveret).
@+11	Ubrugt (reserveret).
@+12	Første byte i CP/M FCB'en.
:	
@+47	Sidste byte i CP/M FCB'en.
@+48	Første byte i filbufferen.
:	
@+175	Sidste byte i filbufferen.

Formatet af flagbyten i @+0 er følgende:

Bit 0..3	Filtype.
Bit 4	Buffer ikke læst fra disk flag.
Bit 5	Buffer ikke skrevet til disk flag eller forudlæst–tegn flag.
Bit 6	Skrivning tilladt flag.
Bit 7	Læsning tilladt flag.

Filtype 0 angiver en diskfil, medens 1 til 5 angiver PolyPascals logiske I/O enheder (CON:, KBD:, LST:, AUX:, og USR:). For typeangivne filer er bit 4 sat hvis bufferens indhold er udefineret, og bit 5 sat hvis data er blevet skrevet i bufferen, men ikke i på disken. For textfiler er bit 5 er sat hvis tegnbufferen i @+3 indeholder et læst men ubrugt

tegn. Bit 6 er sat hvis skrivning er tilladt, og bit 7 er sat hvis læsning er tilladt.

Bufferpointeren indeholder et index (0..127) til bufferen i @+48. For typeangivne og typeløse filer gælder, at de tre ord fra @+4 til @+9 angiver antallet af records i filen, recordlængden i bytes, og det nuværende recordnummer. En typeløs fils FIB har ingen buffer, og bufferpointeren bruges derfor ikke. Længden af en typeløs fils FIB er kun 48 bytes.

Når en textfil arbejder på en logisk enhed, er det kun de to første bytes i FIB'en der anvendes.

23.1.6 Pointere

I PolyPascal–86 består en pointer af to ord (fire bytes). Det første ord indeholder offsetadressen og det andet ord indeholder segmentadressen. NIL lagres som to ord med 0.

I PolyPascal–80 består en pointer af to bytes, der angiver en 16–bits adresse i lageret. Den mindst betydende byte lagres på den laveste adresse. Pointerværdien NIL lagres som adressen 0.

23.2 Datastrukturer

De grundlæggende datatyper kan samles i datastrukturer på tre forskellige måder: Arraystrukturer, poster og diskfiler. Grupperingen berører på ingen måde det interne format af de grundlæggende datatyper. Således har et element i en datastruktur altid det samme interne format som en enkeltvariabel af samme type.

23.2.1 Arraystrukturer

Elementet med den laveste indexværdi lagres på den laveste adresse. Hvis arraystrukturen er flerdimensional, optælles den sidste dimension først.

23.2.2 Poster

Det første felt i en post lagres på den laveste adresse. Hvis posten ikke indeholder en variant–del, er længden givet ved summen af længderne på hver enkelt felt. I modsat fald er længden givet ved længden af den faste del plus længden af mærkefeltet (hvis det eksisterer) plus længden af den største variant. Hver variant starter på den samme lageradresse.

23.2.3 Diskfiler

I modsætning til andre datastrukturer bliver elementerne i en diskfil ikke gemt i lageret men i stedet i en fil på en disk. En diskfil styres via en FIB som beskrevet i afsnit 23.1.5. PolyPascal arbejder med to forskellige diskfiltyper: Textfiler og random access datafiler.

23.2.3.1 Textfiler

En textfil er inddelt i linier. Hver linie består af nul eller flere tegn og afsluttes af en CR/LF sekvens. ASCII værdien af et CR (vogn retur) tegn er 13, og ASCII værdien af et LF (ny linie) tegn er 10. Filen afsluttes af et SUB (Ctrl–Z) tegn. ASCII Værdien af et SUB tegn er 26.

23.2.3.2 Typeangivne filer

En typeangiven fil består af en følge af elementer, der alle er af samme længde og interne format. For at opnå fuld udnyttelse af diskfilens kapacitet, lagres elementerne umiddelbart efter hinanden uafhængigt af sektorgrænser.

I MS–DOS versionen udregnes antallet af elementer i filen fra filens længde, der er lagret i diskettens directory. Da elementlængden hverken gemmes i filen eller i diskettens directory, er det op til programmøren at sikre, at filen altid behandles med samme elementlængde.

I CP/M versionerne af PolyPascal angiver de første fire bytes af en typeangiven fil antallet af elementer i filen og længden af elementerne:

sektor 0, byte 0:	Antal elementer (LSB).
sektor 0, byte 1:	Antal elementer (MSB).
sektor 0, byte 2:	Elementlængde i bytes (LSB).
sektor 0, byte 3:	Elementlængde i bytes (MSB).

Det første elements data følger umiddelbart efter disse kontrolbytes.

23.3 PolyPascal–86 parameteroverførsler

Dette afsnit beskriver de metoder og formater der bruges til at overføre parametre til procedurer og funktioner i PolyPascal–86.

23.3.1 Parametre

PolyPascal–86 overfører parametre via 8086 CPU'ens stak, der adresseres gennem SS:SP registerparret. Ved indgang til en subrutine er returadressen (i kodesegmentet) altid lagret i det øverste ord på stakken. Hvis subrutinen kræver parametre, lagres disse under returadressen (dvs. på højere adresser på stakken), og hvis subrutinen er en funktion, reserveres der plads til funktionsvariablen under parametrene (dvs. på en højere adresse).

De første instruktioner, der udføres af en subrutine, er normalt:

```
PUSH   BP          ;Gem base page register
MOV    BP,SP       ;Peg til ny stak–ramme
SUB    SP,varsize  ;Reserver lokalt arbejdslager
```

hvor 'varsize' er størrelsen af det lokale arbejdslager, der anvendes i subrutinen. Når subrutinen returnerer, udføres:

```
MOV    SP,BP       ;Reset stakpointer
POP    BP          ;Reset base page register
RET    parsize     ;Returner og juster stak
```

hvor 'parsize' er det antal bytes, parametrene optog.

Antag at en EXTERNAL funktion ved navn stak er erklæret ved:

```
FUNCTION stak(VAR i: integer; r: real; s: str16): integer;
```

hvor str16 er typen STRING[16]. Antag envidere at de ovenfor viste indgangsinstruktioner er udført med varsize lig 50 for at reservere 50 bytes lokalt arbejdslager. Stakken vil da se således ud (den øverste linie i figuren svarer til den højeste adresse):

```
001F - 0020   Funktionsvariabel (1 ord).
001D - 001E   i's segmentadresse (1 ord).
001B - 001C   i's offsetadresse (1 ord).
0016 - 001A   r's mantissa (5 bytes).
0015 - 0015   r's exponent (1 byte).
0005 - 0014   Tegnene i s (16 bytes).
0004 - 0004   Længden af s (1 byte).
0002 - 0003   Returadresse (1 ord).
0000 - 0001   Bortgemt base page register (1 ord).
FFCE - FFFF   Lokalt arbejdslager (50 bytes).
```

Hexværdierne er offsetadresser i forhold til base page registeret (BP). Bemærk at der er reserveret plads til funktionsvariablen oven over parametrene, og at parametrene gemmes i "omvendt" rækkefølge.

Funktionsvariable i PolyPascal–86 anvender samme format som almindelige variable. Det samme gælder parametre, bortset fra, at integers, booleans, tegn og selvdefinerede skalarer, der overføres som value–parametre, altid optager et ord (2 bytes), selvom deres værdi godt kan udtrykkes i en enkelt byte. Var–parametre overføres altid som pointere, dvs. to ord, der indeholder en offsetadresse og en segmentadresse. De formater, der beskrives i afsnit 23.4, gælder ikke for PolyPascal–86.

Nedenfor vises et eksempel på en EXTERNAL subrutine (en funktion), der konverterer et tegn til et stort bogstav. Funktionen erklæres på følgende måde i PolyPascal programmet:

```
FUNCTION upcase(ch: char): char; EXTERNAL 'UPCASE';
```

Funktionen er skrevet i 8086 assembler, og oversat til maskinkode ved hjælp af operativsystemets assembleroversætter. Kildeteksten er vist nedenfor:

```
UPCASE: PUSH  BP        ;Gem BP
        MOV   BP,SP     ;Peg til stak–ramme
        MOV   AL,4[BP]  ;Hent ch parameter
        CMP   AL,'a'    ;Mindre end 'a'?
        JB    UPC1      ;Ja => hop
        CMP   AL,'z'    ;Større end 'z'?
        JA    UPC1      ;Ja => hop
        SUB   AL,20H    ;Konverter til stort
UPC1:   POP   BP        ;Reset BP
        RET   3         ;Returner og juster stak
```

Bemærk at der poppes tre bytes af stakken ved udgang – et ord til ch parametren og en byte til funktionsvariablen (der ikke anvendes i dette tilfælde).

23.3.2 Funktionsresultater

I PolyPascal–86 returneres værdier af alle skalare typer (undtagen real) i AX registeret. Hvis resultatet kun kræver en byte, skal AH sættes til nul. Bemærk, at boolean funktioner skal udføre en OR AX,AX in-

struktion umiddelbart før de returnerer, for at sikre, at CPU'ens Z flag afspejler værdien i AX. Funktionen skal fjerne alle parametre og funktionsvariablen fra stakken før eller når den returnerer.

Reals returneres på toppen af stakken med formatet givet i 23.1.2. Dette kan nemt opnås ved at lade funktionsvariablen blive liggende på stakken når der returneres. Funktionen skal således kun fjerne parametrene.

Strenge er en smule mere komplicerede. Før funktionen returnerer skal den fjerne alle parametrene og normalisere strengresultatet, dvs. flytte strengen op på en højere adresse, således at den optager det antal bytes, der svarer til dens dynamiske længde plus en. Ved udgang skal stakpointeren pege på den byte, der indeholder strengens længde.

Pointerværdier returneres i DX:AX registerparret (segmentadresse i DX og offsetadresse i AX). Funktionen skal fjerne alle parametre samt funktionsvariablen når den returnerer.

23.4 PolyPascal–80 parameteroverførsler

Dette afsnit beskriver de formater og metoder der bruges til at overføre parametre til procedurer og funktioner i PolyPascal–80.

Parametre overføres til procedurer og funktioner via Z–80 processorens stak. Normalt har dette ingen betydning for programmøren, da PolyPascal sørger for både at PUSHe parametrene før et kald og POPpe dem i begyndelsen af underprogrammet. Skulle programmøren imidlertid ønske at kalde eksterne maskinkoderutiner fra et Pascalprogram, skal disse underprogrammer skrives sådan, at de selv sørger for at POPpe parametrene fra stakken.

Ved indgang til et eksternt underprogram ligger returadressen øverst på stakken. Eventuelle parametre findes "under" returadressen (dvs. på højere adresser i lageret). For at få adgang til parametrene må returadressen POPpes af. Derefter kan parametrene POPpes, og til sidst skal returadressen PUSHes tilbage.

23.4.1 Var–parametre

En var–parameter overføres på stakken som et 16–bits tal, der angiver adressen på den første byte, der optages af den aktuelle parameter.

23.4.2 Value–parametre

Hvis en parameter er en value–parameter, er formatet af de bytes der overføres afhængigt af parametrens type.

23.4.2.1 Skalarer

Alle skalarer, undtagen reals, dvs. integers, booleans, tegn (char), selvdefinerede skalarer, og delintervaller overføres på stakken som 16–bits ord. Hvis værdien er af en type, der normalt kun optager en byte, er den mest betydende byte nul. Normalt POPpes et 16–bits ord af stakken med en POP HL, POP DE eller POP BC instruktion.

23.4.2.2 Reals

En real overføres på stakken som tre 16–bits ord. Hvis disse ord POPpes med den følgende instruktionssekvens:

```
POP   HL
POP   DE
POP   BC
```

vil L indeholde exponenten, H mantissens femte byte (mindst betydende), E den fjerde byte, D den tredje byte, C den anden byte, og B den første byte (mest betydende).

23.4.2.3 Strenge

Når en streng er placeret øverst på stakken, indeholder den byte, der adresseres af SP registeret, længden af strengen, og bytene fra SP+1 til SP+n (hvor n er længden af strengen) indeholder strengens tegn. Den følgende instruktionssekvens POPper en streng af stakken og lagrer den i STRBUF:

```
LD    DE,STRBUF
LD    HL,0
LD    B,H
ADD   HL,SP
LD    C,(HL)
INC   BC
LDIR
LD    SP,HL
```

23.4.2.4 Mængder

En mængde optager altid 32 bytes på stakken (mængder komprimeres kun når de lagres). Den følgende instruktionssekvens POPper en mængde af stakken og lagrer den i SETBUF.

```
LD      DE,SETBUF
LD      HL,0
ADD     HL,SP
LD      BC,32
LDIR
LD      SP,HL
```

Den mindst betydende byte i mængden lagres på den laveste adresse i STRBUF.

23.4.2.5 Pointere

En pointer overføres på stakken som et 16–bits ord, der angiver en adresse i lageret. Pointerværdien NIL svarer til adressen 0.

23.4.2.6 Arraystrukturer og poster

Arraystrukturer og poster bliver faktisk ikke PUSHet på stakken, selvom de overføres som value–parametre. I stedet overføres et 16–bits ord, der angiver adressen på den første byte, der optages af parameteren. Herefter er det op til programmøren at POPpe denne adresse og anvende den som kildeadresse i en blokkopiering.

23.4.3 Funktionsresultater

Brugerdefinerede eksterne funktioner bør nøje overholde de nedenfor angivne regler, når de returnerer resultater.

Værdier af alle skalare typer, undtagen reals, returneres i HL registerparret. Hvis resultatet er af en type, der normalt kun fylder en byte, skal L indeholde resultatet og H indeholde 0.

Reals returneres i BC, DE, og HL registerparrene. B, C, D, E og H skal indeholde mantissen (mest betydende byte i B), og L skal indeholde exponenten.

Strenge returneres på toppen af stakken, som beskrevet i afsnit 23.4.2.3.

Pointere returneres i HL registerparret.

24. Lagerorganisering

24.1 PolyPascal–86 lagerorganisering

Når et program, der er oversat af PolyPascal–86, udføres (enten som en '.CMD' fil i CP/M–86 eller som en '.COM' fil i MS–DOS eller fra en RUN kommando i PolyPascal), bliver der allokeret tre segmenter til programmet: Et kodesegment, et datasegment og et staksegment.

MS–DOS kodesegment (CS er kodesegmentregisteret):

CS:0000	–	CS:00FF	MS–DOS base page.
CS:0100	–	CS:EOFR	Run–time programdel.
CS:EOFR	–	CS:EOFP	Programkode.
CS:EOFP	–	CS:EOFC	Ubrugt.

MS–DOS datasegment (DS er datasegmentregisteret):

DS:0000	–	DS:EOFW	Run–time programdelens arbejdslager.
DS:EOFW	–	DS:EOFM	Hovedprogrammets variable.
DS:EOFM	–	DS:EOFD	Ubrugt.

CP/M–86 kodesegment (CS er kodesegmentregisteret):

CS:0000	–	CS:EOFR	Run–time programdel.
CS:EOFR	–	CS:EOFP	Programkode.
CS:EOFP	–	CS:EOFC	Ubrugt.

CP/M–86 datasegment (DS er datasegmentregisteret):

DS:0000	–	DS:00FF	CP/M–86 base page.
DS:0100	–	DS:EOFW	Run–time programdelens arbejdslager.
DS:EOFW	–	DS:EOFM	Hovedprogrammets variable.
DS:EOFM	–	DS:EOFD	Ubrugt.

De ubrugte områder (CS:EOFP–CS:EOFC og DS:EOFM–DS:EOFD) allokeres kun hvis der blev specificeret en minimum segmentstørrelse, der er større end den nødvendige størrelse, i den PROGRAM kommando, der oversatte programmet. Kode og datasegmenterne er aldrig større end 64K bytes hver.

Staksegmentet er mere kompliceret, da det kan være større end 64K bytes. Ved start af programmet sættes staksegmentregisteret (SS) og stakpointeren (SP) således, at SS:SP peger på den sidste byte i staksegmentet. SS ændres aldrig under udførelsen af programmet, men SP arbejder sig nedefter, indtil bunden af staksegmentet nås, eller indtil SP bliver nul (svarende til en stakstørrelse på 64K bytes).

Heapen vokser fra bunden af staksegmentet op mod selve stakken, der ligger i toppen. Når en variabel allokeres på heapen, bliver heap–pointeren (en pointer, der vedligeholdes af PolyPascal run–time programdelen) flyttet opad og derefter normaliseret, således at dens offsetadresse altid er mellem $0000 og $000F. Den maksimale størrelse af en dynamisk variabel der kan allokeres på heapen er derfor 65521 bytes, svarende til $10000 minus $000F. Den sammenlagte størrelse af variablene på heapen kan imidlertid godt overstige 64K bytes.

Heap–pointeren kan adresseres af programmøren ved hjælp af hptr standard identifieren. hptr er en typeløs pointer, der er kompatibel med alle pointertyper, på samme måde som NIL er det. Tilskrivninger til hptr bør kun foretages med yderste forsigtighed.

24.2 PolyPascal–80 lagerorganisering

Under oversættelsen af et program er systemets lager organiseret som vist nedenfor:

0000	–	00FF	CP/M og run–time arbejdslager
0100	–	EOFR	Run–time programdel
EOFR	–	EOFC	PolyPascal monitor, editor og compiler
EOFC	–	EOFW	Compilerens arbejdslager
EOFW	–	EOFM	Fejlmeddelelser (PPAS.ERM)
EOFM	–	EOFT	Kildetekst
EOFT	–	>>>>	Objektkode
<<<<	–	MTOP	Compilerens symboltabel
<<<<	–	LTOP	CPU stak
LTOP	–	FFFF	CP/M operativsystem

Hvis PPAS.ERM filen ikke blev indlæst ved start af PolyPascal systemet, starter kildeteksten ved EOFW. Når compileren startes fra en COMPILE eller en RUN kommando, bliver objektkoden gemt direkte ud i lageret umiddelbart efter kildeteksten. CPU stakken arbejder sig nedefter startende fra LTOP (slutadressen på CP/M TPA'en). Symboltabellen arbejder sig nedefter startende fra MTOP, der ligger 1K bytes under LTOP (LTOP–$400).

Under kørsel af et program, der blev startet med RUN kommandoen, er systemets lager organiseret som vist nedenfor:

0000	–	00FF	CP/M og run–time arbejdslager
0100	–	EOFR	Run–time programdel
EOFR	–	EOFC	PolyPascal monitor, editor og compiler
EOFC	–	EOFW	Compilerens arbejdslager
EOFW	–	EOFM	Fejlmeddelelser (PPAS.ERM)
EOFM	–	EOFT	Kildetekst
EOFT	–	EOFP	Objektkode
EOFP	–	BOFH	Ubrugt
BOFH	–	>>>>	Heap (styres via hptr)
<<<<	–	BOFR	Procedurestak (styres via rptr)
<<<<	–	BOFS	CPU stak (styres via sptr)
BOFS	–	FTOP	Ubrugt
FTOP	–	LTOP	Programmets variable
LTOP	–	FFFF	CP/M operativsystem

EOFP er slutadressen på objektkoden, og hptr (heap–pointeren) sættes til denne adresse ved start af programmet (BOFH=EOFP). Området mellem FTOP og LTOP bruges til programmets variable. FTOP er slutadressen på det frie lager, og sptr (CPU stakpointeren) sættes til denne adresse ved start af programmet (BOFS=FTOP). Procedurestakken bruges kun af rekursive procedurer og funktioner (til at gemme kopier af deres arbejdslager). Ved begyndelsen af et program sættes rptr (procedurestakpointeren) til at pege 1K bytes under CPU stakpointeren (BOFR=BOFS–$400).

Under kørsel af en programfil er lageret organiseret som vist nedenfor:

0000	-	00FF	CP/M og run-time arbejdslager
0100	-	EOFR	Run-time programdel
EOFR	-	SOFP	Ubrugt (brugerdefinerede maskinkoderutiner)
SOFP	-	EOFP	Objektkode
EOFP	-	BOFH	Ubrugt
BOFH	-	>>>>	Heap (styres via hptr)
<<<<	-	BOFR	Procedurestak (styres via rptr)
<<<<	-	BOFS	CPU stak (styres via sptr)
BOFS	-	FTOP	Ubrugt
FTOP	-	PTOP	Programmets variable
PTOP	-	LTOP	Ubrugt
LTOP	-	FFFF	CP/M operativsystem

SOFP er startadressen på objektkoden, svarende til <start> parametren i PROGRAM og OBJECT kommandoerne. PTOP er slutadressen på programmets arbejdslager, svarende til <slut> parametren i PROGRAM og OBJECT kommandoerne.

Som det fremgår af de ovenfor viste memory maps, eksisterer der tre stak-lignende strukturer i lageret under udførelsen af et program: Heapen, CPU stakken og procedurestakken.

Heapen bruges til at lagre dynamiske variable, og den styres via new, mark og release procedurerne. Ved begyndelsen af et program sættes heap-pointeren (hptr) til startadressen på det frie lager.

CPU stakken bruges til at gemme mellemresultater under udregning af udtryk, og til at overføre parametre til underprogrammer. Desuden bruger en aktiv FOR sætning to bytes på CPU stakken. Ved begyndelsen af et program sættes CPU stakpointeren (sptr) til slutadressen på det frie lager.

Procedurestakken anvendes udelukkende af rekursive procedurer og funktioner (dvs. procedurer og funktioner, der er oversat i {$A+} stillingen). Ved indgang til en rekursiv procedure eller funktion bliver hele underprogrammets arbejdslager kopieret ned på procedurestakken, og umiddelbart før underprogrammet returnerer, kopieres det tilbage igen. Ved begyndelsen af et program sættes procedurestakpointeren (rptr) til at pege 1K bytes ($400) under CPU stakpointeren.

Ved hjælp af tre prædefinerede variable er det muligt for programmøren at flytte på heapen og stakkene:

hptr	Heap–pointeren.
rptr	Procedurestakpointeren.
sptr	CPU stakpointeren.

Typen af disse variable er integer. Bemærk, at variablene kun må anvendes i udtryk og i tilskrivninger; de kan altså ikke bruges som var–parametre til underprogrammer.

Når der rettes på heap– og/eller stakpointerne er det programmørens opgave at sikre, at den følgende regel overholdes:

hptr < rptr < sptr

Hvis denne relation ikke er sand, er følgerne uforudsigelige (og til tider ganske katastrofale for programmet). Det er naturligvis ikke tilladt at ændre på heap– og stakpointervariablene, når den givne struktur først er taget i anvendelse. Derfor bør tilskrivninger til disse variable altid foretages i begyndelsen af et program.

Hvergang new proceduren kaldes, og hvergang der udføres et kald til et rekursivt underprogram, tester systemet, at der ikke er sket en kollision mellem heapen og procedurestakken, altså at hptr er mindre end rptr. Hvis dette ikke er tilfældet, opstår en kørselsfejl.

Bemærk, at der på intet tidspunkt testes for, om CPU stakken har fået overløb ind i "bunden" af procedurestakken. Denne situation opstår først, når rekursive procedurekald nestes en 300–500 gange, hvilket sker meget sjældent. Skulle det imidlertid være påkrævet at neste så "dybt", må programmet flytte procedurestakpointeren ned i lageret for at skabe mere plads. Dette gøres med sætningen:

```
rptr:=sptr-2*maxdybde-512;
```

hvor maxdybde er det højeste nestingniveau, der kan forekomme. De 512 bytes (eller deromkring), der reserveres derudover, sikrer, at der er plads til mellemresultater under udregning af udtryk.

25. Interruptstyring

Den kode, der genereres af PolyPascal, såvel som run–time programdelen, er fuldt ud interruptbar. Hvis man ønsker det, kan interruptrutiner skrives i PolyPascal.

25.1 PolyPascal–86 interruptprocedurer

En PolyPascal–86 interruptprocedure skal selv sørge for at bevare registrene AX, BX, CX, DX, SI, DI, DS og ES. Dette opnås ved at placere følgende CODE sætning i begyndelsen af proceduren:

```
CODE $50,$53,$51,$52,$56,$57,$1E,$06,$FB;
```

Den sidste byte ($FB) er en STI instruktion, der tillader yderligere interrupts – i nogle tilfælde er dette nødvendigt, i andre ikke. Den følgende CODE sætning skal være den sidste sætning i proceduren:

```
CODE $07,$1F,$5F,$5E,$5A,$59,$5B,$58,$8B,$E5,$5D,$CF;
```

Den sidste byte ($CF) er en IRET instruktion, der udføres i stedet for den RET instruktion, der genereres af compileren.

En interruptprocedure må ikke udføre I/O operationer ved hjælp af standardprocedurer og funktioner, og ej heller operationer på real variable, da disse dele af PolyPascal run–time programdelen ikke er re–entrante. Ydermere må en interruptprocedure ikke anvende WITH sætninger.

Hvis en interruptprocedure ønsker adgang til et programs globale variable, skal proceduren selv initialisere DS registeret, da det ikke er sikkert, at DS peger på programmets datasegment, når et interrupt forekommer. Førend interrupts kan tillades, skal hovedprogrammet lagre indholdet af DS på et kendt sted, for eksempel i kodesegmentet eller i en ubrugt interruptvektor. Ved indgang skal interruptproceduren læse DS fra denne adresse.

For at reservere et ord i kodesegmentet, hvor indholdet af DS kan gemmes, kan følgende erklæring bruges:

```
CONST
  sdseg: integer = 0;
```

Følgende CODE sætning vil gemme indholdet af DS i dette ord:

```
sdseg:=dseg;
```

Dette kræver selvfølgelig at koden ikke er lagt i ROM lager.

Interrupt rutinen kan nu reetablere DS–registeret med følgende CODE sætning:

```
CODE $2E,$8E,$1E,sdseg;
```

Det nedenfor viste program anvender en interruptprocedure. Det antages, at der hvert sekund forekommer en interrupt til vektor 192 ($C0). Ved programstarten lagres indholdet af DS registeret som ovenfor beskrevet. Programmet initialiserer herefter interruptvektor 192 og "enabler" interrupts. Når der forekommer en interrupt 192 bliver kontrollen overført til "plustid" proceduren. Den gemmer registrene AX, BX, CX, DX, SI, DS og ES, og læser derefter DS registeret fra konstanten. Nu da DS er initialiseret, kan proceduren behandle de globale variable. Ved udgang hentes alle registre fra stakken, også DS, SP og BP, hvorefter en IRET instruktion udføres. Ved udskriften aflæses tiden, medens interrupts er afbrudt. Undlades CLI instruktionen, risikerer man at tallene vil ændre sig, medens de blive aflæst. Inden hovedprogrammet slutter, udfører det en CLI instruktion for at afbryde yderligere interrupts.

```
PROGRAM interrupt;
CONST
  sdseg:integer = 0;
VAR
  int_vekt: ^integer AT $0000:$0300;
  timer,minutter,sekunder,t,m,s: integer;

PROCEDURE plustid;
BEGIN
  CODE $50,$53,$51,$52,$56,$57,$1E,$06,$2E,$8E,$1E,sdseg;
  sekunder:=succ(sekunder);
```

```
    IF sekunder=60 THEN
    BEGIN
      sekunder:=0; minutter:=succ(minutter);
      IF minutter=60 THEN
      BEGIN
        minutter:=0; timer:=succ(timer) mod 24;
      END;
    END;
    CODE $07,$1F,$5F,$5E,$5A,$59,$5B,$58,$8B,$E5,$5D,$CF;
  END;

  BEGIN
    sdseg:=dseg; int_vekt:=ptr(cseg,ofs(plustid));
    write('Skriv tid (time minut sekund): ');
    readln(timer,minutter,sekunder);
    REPEAT
      CODE $FA;
      t:=timer; m:=minutter; s:=sekunder;
      CODE $FB;
      writeln('Klokken er: ',t:2,':',m:2,':',s:2);
    UNTIL keypress;
    CODE $FA;
  END.
```

25.2 PolyPascal–80 interruptprocedurer

Interruptprocedurer skal altid oversættes som absolutte procedurer (i {$A+} stillingen), de må ikke tage parametre, og de skal selv sørge for, at alle registre, der anvendes, bliver bevaret. Det sidstnævnte opnås ved, at placere en CODE sætning, der indeholder de nødvendige PUSH instruktioner, i begyndelsen af proceduren, og en anden CODE sætning, der indeholder de tilsvarende POP instruktioner, i slutningen af proceduren. Desuden bør den sidste instruktion i den afsluttende CODE sætning være en EI instruktion ($FB), der aktiverer yderligere interrupts. Hvis hardwaren anvender "daisy–chainede" interrupts, kan den afsluttende CODE sætning envidere angive en RETI instruktion ($ED,$4D), der så udføres i stedet for den RET instruktion, der genereres af compileren.

De generelle regler for hvilke registre, der anvendes af den kode, der genereres af PolyPascal, er, at operationer på skalarer (undtagen reals) kun benytter AF, BC, DE og HL, at andre operationer desuden benytter IX og IY, og at operationer på reals tillige benytter de alternative registre.

En interruptprocedure må ikke udføre I/O operationer ved hjælp af standardprocedurer og funktioner, da disse rutiner ikke er re–entrante. Bemærk desuden, at kald til BDOS'en (og eventuelt også BIOS'en, afhængigt af den aktuelle implementation af CP/M) bør undgås i interruptprocedurer, da disse rutiner ej heller er re–entrante.

Programmøren kan aktivere og passivere interrupts fra et program ved hjælp af EI og DI instruktioner, der indsættes med CODE sætninger.

Hvis der anvendes mode 0 (IM 0) eller mode 1 (IM 1) interrupts, er det programmørens opgave at initialisere "restart" områderne. Bemærk, at RST 0 interrupts ikke kan anvendes, da RST 0 området fra adresse 0 til 7 bruges af CP/M operativsystemet. Det samme gælder ikke–maskerbare interrupts (NMI'er), da området omkring adresse $66 er en del af CP/M's arbejdslager. Hvis der anvendes mode 2 interrupts, er det programmørens opgave at erklære en adressetabel (et ARRAY OF integer) på en fast adresse og initialisere den, samt at initialisere I registeret gennem en CODE sætning i begyndelsen af programmet.

Det nedenfor viste program anvender en interruptprocedure. Det antages, at der hvert sekund forekommer en mode 1 interrupt (et kald til adresse $38). Programmet gælder kun for PolyPascal–80:

```
PROGRAM interrupt; {$A+}
VAR
  rsthop: byte AT $38;
  rstadr: integer AT $39;
  timer,minutter,sekunder,t,m,s: integer;

PROCEDURE plustid;
BEGIN
  CODE $F5,$E5,$D5,$C5;
  sekunder:=succ(sekunder);
  IF sekunder=60 THEN
  BEGIN
    sekunder:=0; minutter:=succ(minutter);
    IF minutter=60 THEN
    BEGIN
      minutter:=0; timer:=succ(timer);
      IF timer=24 THEN timer:=0;
    END;
  END;
  CODE $C1,$D1,$E1,$F1,$FB;
END;
```

```
BEGIN
  rsthop:=$C3; rstadr:=addr(plustid);
  write('Skriv tid (time minut sekund): ');
  readln(timer,minutter,sekunder);
  CODE $ED,$56;
  REPEAT
    CODE $F3;
    t:=timer; m:=minutter; s:=sekunder;
    CODE $FB;
    writeln('Klokken er: ',t:2,':',m:2,':',s:2);
  UNTIL keypress;
  CODE $F3;
END.
```

Da plustid interruptproceduren kun udfører heltalsoperationer, behøver den kun gemme AF, BC, DE og HL registerparrene. CODE sætningen i slutningen af plustid angiver både de nødvendige POP instruktioner og en EI instruktion, der akiverer yderligere interrupts. I begyndelsen af programmet klargøres en hopvektor til plustid rutinen i adresse $38. Derefter indlæses det nuværende klokkeslet, interrupt mode 1 vælges med en IM 1 instruktion ($ED,$56), og til sidst aktiveres interrupts med en EI instruktion ($FB). En DI instruktion ($F3) bruges til at passivere interrupts umiddelbart før programmet slutter.

26. Forskelle mellem PolyPascal og Standard Pascal

PolyPascal følger nøje Jensen & Wirth's definition af Standard Pascal, der gives i bogen "Pascal User Manual and Report". Der er imidlertid enkelte mindre forskelle, og disse beskrives nedenfor. Bemærk, at dette kapitel ikke omhandler de udvidelser, der findes i PolyPascal.

Dynamiske variable

Dynamiske variable oprettes og fjernes med new, mark og release procedurerne i stedet for med de af standarden foreslåede new og dispose procedurer. Dette skyldes til dels ønsket om at sikre kompatibilitet med andre Pascal compilere (f.eks. UCSD Pascal), men primært er grunden, at de nye procedurer er mere effektive, hvad angår kørselshastighed og kodestørrelse. Desuden tillader new proceduren i PolyPascal ikke angivelser af varianter (dette kan relativt nemt omgås ved at bruge allocate proceduren).

Get og put procedurerne

get og put standardprocedurerne findes ikke i PolyPascal. I stedet er read og write procedurerne udvidet, så de nu kan anvendes til alle typer af input og output. Der er tre grunde til dette: For det første er read og write lettere at bruge og nemmere at forstå, for det andet er de hurtigere, og for det tredje kan størrelsen af filvariable reduceres, når der ikke er behov for buffer–variable.

GOTO sætninger

En GOTO sætning må ikke hoppe til en label, der ligger udenfor den nuværende blok (som regel er dette også dårlig programmeringsteknik).

Page proceduren

page standardproceduren findes ikke i PolyPascal, da der i CP/M og MS–DOS operativsystemerne ikke findes en standard for, hvilket kontroltegn, der skal anvendes til sideskift.

Procedurer og funktioner som parametre

PolyPascal tillader ikke, at procedurer og funktioner anvendes som parametre. Der er to grunde hertil: For det første er definitionen af Standard Pascal meget uklar på dette punkt og for det andet bruges denne facilitet næsten aldrig.

Pakkede variable

Det reserverede ord PACKED har ingen betydning i PolyPascal (men tillades dog). Variable pakkes i stedet automatisk, når det er muligt, og som følge heraf, er pack og unpack procedurerne ikke implementerede.

Appendix A
Oversigt over standard procedurer og funktioner

Dette appendix giver en oversigt over PolyPascal's standard procedurer og funktioner. De følgende notationer anvendes:

<type>	Angiver enhver type.
<string>	Angiver enhver strengtype.
<file>	Angiver enhver filtype.
<scalar>	Angiver enhver skalar type (undtagen real).
<pointer>	Angiver enhver pointertype.

Bemærk, at visse procedurer og funktioner tillader var–parametre af enhver type. I så tilfælde angives der ingen type for disse parametre. Følgende forkortelser kan eventuelt forekomme i margenen ved beskrivelsen af en procedure eller en funktion:

86	Gælder kun PolyPascal–86.
80	Gælder kun PolyPascal–80.
MS	Gælder kun MS–DOS versionen af PolyPascal–86.
CP	Gælder kun CP/M versionen af PolyPascal–86.

Input/Output rutiner

De nedenfor viste procedurer benytter en non–standard syntaks i deres parameterlister.

```
PROC      read  (VAR f: FILE OF <type>; VAR v: <type>);
          read  (VAR f: text; VAR i: integer);
          read  (VAR f: text; VAR r: real);
          read  (VAR f: text; VAR c: char);
          read  (VAR f: text; VAR s: <string>);
PROC    readln  (VAR f: text);
PROC     write  (VAR f: FILE OF <type>; VAR v: <type>);
          write  (VAR f: text; i: integer);
          write  (VAR f: text; r: real);
          write  (VAR f: text; b: boolean);
          write  (VAR f: text; c: char);
          write  (VAR f: text; s: <string>);
PROC    writeln  (VAR f: text);
```

Filbehandlingsrutiner

PROC	append	(VAR f: text);	MS
PROC	assign	(VAR f: <file>; name: <string>);	
PROC	blockread	(VAR f: FILE; VAR dest; numrec: integer; VAR numread: integer);	
PROC	blockwrite	(VAR f: FILE; VAR dest; numrec: integer; VAR numwrit: integer);	
PROC	chain	(VAR f: <file>);	
PROC	chdir	(path: <string>);	MS
PROC	close	(VAR f: <file>);	
FUNC	eof	(VAR f: <file>): boolean;	
FUNC	eoln	(VAR f: text): boolean;	
PROC	erase	(VAR f: <file>);	
PROC	execute	(VAR f: <file>);	
PROC	flush	(VAR f: FILE OF <type>);	
	flush	(VAR f: text);	MS
PROC	gcdir	(drive: integer; VAR path: <string>);	MS
FUNC	length	(VAR f: FILE OF <type>): integer;	
	length	(VAR f: FILE): integer;	
FUNC	longlen	(VAR f: FILE OF <type>): real;	MS
	longlen	(VAR f: FILE): real;	MS
FUNC	longpos	(VAR f: FILE OF <type>): real;	MS
	longpos	(VAR f: FILE): real;	MS
PROC	mkdir	(path: <string>);	MS
FUNC	position	(VAR f: FILE OF <type>): integer;	
	position	(VAR f: FILE): integer;	
PROC	rename	(VAR f: <file>; name: <string>);	
PROC	reset	(VAR f: <file>);	
	reset	(VAR f: FILE; reclen: integer);	MS
PROC	rewrite	(VAR f: <file>);	
	rewrite	(VAR f: FILE; reclen: integer);	MS
PROC	rmdir	(path: <string>);	MS
PROC	seek	(VAR f: FILE OF <type>; pos: integer);	
	seek	(VAR f: FILE; pos: integer);	
	seek	(VAR f: FILE OF <type>; pos: real);	MS
	seek	(VAR f: FILE; pos: real);	MS
FUNC	seof	(VAR f: <file>): boolean;	
FUNC	seoln	(VAR f: text): boolean;	
PROC	truncate	(VAR f: <file>);	MS

Aritmetiske rutiner

```
FUNC        abs (i: integer): integer;
            abs (r: real): real;
FUNC      arctan (r: real): real;
FUNC        cos (r: real): real;
FUNC        exp (r: real): real;
FUNC       frac (r: real): real;
FUNC        int (r: real): real;
FUNC         ln (r: real): real;
FUNC        sin (r: real): real;
FUNC        sqr (i: integer): integer;
            sqr (r: real): real;
FUNC       sqrt (r: real): real;
```

Skalare rutiner

```
FUNC        odd (i: integer): boolean;
FUNC       pred (x: <scalar>): <scalar>;
FUNC       succ (x: <scalar>): <scalar>;
```

Konverteringsrutiner

```
FUNC        chr (i: integer): char;
FUNC        ord (x: <scalar>): integer;
FUNC      round (r: real): integer;
FUNC      trunc (r: real): integer;
```

Strengbehandlingsrutiner

Bemærk, at str proceduren benytter en non–standard syntaks til den numeriske parameter. Bemærk også, at fmt funktionen, der kun findes i Business versionen, kan tage et variabelt antal parametrer.

```
FUNC      concat (s1,s2,...,sn: <string>): <string>;
FUNC        copy (s: <string>; pos,len: integer): <string>;
PROC      delete (VAR s: <string>; pos,len: integer);
FUNC         fmt (image: <string>; r: real): <string>;
             fmt (image: <string>; s: <string>): <string>;
PROC      insert (s: <string>; VAR d: <string>; pos: integer);
FUNC         len (s: <string>): integer;
FUNC         pos (pattern,source: <string>): integer;
PROC         str (i: integer; VAR s: <string>);
             str (r: real; VAR s: <string>);
PROC         val (s: <string>; VAR i,p: integer);
             val (s: <string>; VAR r: real; VAR p: integer);
```

Heap og pointerrutiner

```
FUNC        addr (VAR variable): <pointer>;                  86
PROC    allocate (VAR p: <pointer>; size: integer);
PROC        mark (VAR p: <pointer>);
FUNC    memavail : integer;
PROC         new (VAR p: <pointer>);
FUNC         ord (p: <pointer>): integer;
FUNC         ptr (i: integer): <pointer>;                     80
             ptr (seg,ofs: integer): <pointer>;               86
PROC     release (VAR p: <pointer>);
```

Overlay rutiner

```
PROC     ovdrive (drive: integer);                            CP
PROC      ovpath (path: <string>);                            MS
```

Diverse rutiner

FUNC	addr	(VAR variable): integer;	80
	addr	(<procedure identifier>): integer;	80
	addr	(<function identifier>): integer;	80
PROC	bdos	(func,param: integer);	80
PROC	bios	(func,param: integer);	80
FUNC	bdos	(func,param: integer): integer;	80
FUNC	bios	(func,param: integer): integer;	80
FUNC	bdosb	(func,param: integer): byte;	80
FUNC	biosb	(func,param: integer): byte;	80
FUNC	cseg	: integer;	86
FUNC	dseg	: integer;	86
PROC	exit	;	
PROC	fill	(VAR dest; length: integer; data: byte);	
	fill	(VAR dest; length: integer; data: char);	
PROC	gotoxy	(x,y: integer);	
PROC	halt	;	
FUNC	hi	(i: integer): integer;	
FUNC	iores	: integer;	
FUNC	keypress	: boolean;	
FUNC	lo	(i: integer): integer;	
PROC	move	(VAR source,dest; length: integer);	
FUNC	ofs	(VAR variable): integer;	86
	ofs	(<procedure identifier>): integer;	86
	ofs	(<function identifier>): integer;	86
FUNC	pwrten	(exp: integer): real;	
FUNC	random	(range: integer): integer;	
	random	: real;	
PROC	randomize	;	
FUNC	seg	(VAR variable): integer;	86
FUNC	size	(VAR variable): integer;	
	size	(<type identifier>): integer;	
FUNC	sseg	: integer;	86
FUNC	swap	(i: integer): integer;	
PROC	swint	(<constant byte>; VAR regpack);	86

Appendix B
Oversigt over operatorer

Dette appendix giver en oversigt over alle de operatorer, der findes i PolyPascal. Operatorerne er grupperet efter prioritet, og grupperne er opskrevet efter faldende prioritet.

Operator	**Operation**	**Operandtype(r)**	**Resultattype(r)**
+ alene	ingen	integer, real	som operand
– alene	negation	integer, real	som operand
NOT	komplementering	integer, boolean	som operand
*	multiplikation	integer, real	integer, real
	foreningsmængde	mængde	som operand
/	division	integer, real	real
DIV	heltalsdivision	integer	integer
MOD	modulus	integer	integer
AND	logisk AND	integer, boolean	som operand
SHL	venstreskift	integer	integer
SHR	højreskift	integer	integer
+	addition	integer, real	integer, real
	sammensætning	streng	som operand
	fællesmængde	mængde	som operand
–	subtraktion	integer, real	integer, real
	differensmængde	mængde	som operand
OR	logisk OR	integer, boolean	som operand
EXOR	logisk EXOR	integer, boolean	som operand

=	lighed	skalar	boolean
	lighed	streng	boolean
	lighed	mængde	boolean
	lighed	pointer	boolean
<>	ulighed	skalar	boolean
	ulighed	streng	boolean
	ulighed	mængde	boolean
	lighed	pointer	boolean
>=	større eller lig	skalar	boolean
	større eller lig	streng	boolean
	delmængde–test	mængde	boolean
<=	mindre eller lig	skalar	boolean
	mindre eller lig	streng	boolean
	delmængde–test	mængde	boolean
>	større end	skalar	boolean
	større end	streng	boolean
<	mindre end	skalar	boolean
	mindre end	streng	boolean
IN	element–test	se nedenfor	boolean

IN operatorens første operand kan være af enhver skalar type (undtagen real) og den anden operand skal være en mængde af samme type.

Appendix C
Oversigt over compilerdirektiver

Et compilerdirektiv skrives i en kommentar, og kan derfor anvendes når en kommentar er tilladt. Et $ tegn som det første tegn i en kommentar angiver, at kommentaren indeholder et eller flere compilerdirektiver. Det generelle format af en kommentar, der indeholder compilerdirektiver, er:

{$<compilerdirektiver> <kommentar>}

eller

(*$<compilerdirektiver> <kommentar>*)

Compilerdirektiverne angives som en liste af instruktioner, adskilt af kommaer, og hver instruktion begynder med et bogstav. Hvis bogstavet udvælger et compilerflag, skal det efterfølges af et plus (+) eller et minus (–), alt efter om den funktion, der styres af flaget, skal aktiveres eller deaktiveres. Hvis bogstavet udvælger et compilerregister (se F, G, P og W nedenfor), skal det efterfølges af et tal eller et bogstav, der angiver registerets nye værdi, og hvis bogstavet udvælger en speciel facilitet (se I nedenfor), skal det efterfølges af et eller flere tegn, der fungerer som parameter til denne facilitet. Nogle eksempler på compilerdirektiver:

{$R–} {$S+,I+,A–} {$W6,B–} {$I MAX.LIB}

De følgende compilerdirektiver er tilgængelige:

A Kun PolyPascal–80. Når dette compilerflag er aktiveret, instruerer det compileren om, at generere absolut kode til procedurer og funktioner, dvs. kode, der ikke tillader rekursion, men som kører hurtigere og fylder mindre end den modsvarende rekursive kode. Yderligere detaljer findes i afsnit 15.6.

Værdi ved start af compiler: A–

B Når dette compilerflag er aktivt ved begyndelsen af en programblok, da vil alle kald til read og readln uden filvariabel altid medføre, at en ny linie indlæses. Ellers bliver en ny linie først indlæst når liniebufferen er blevet tom. Yderligere detaljer herom findes i afsnit 16.1.

Værdi ved start af compiler: B+

C Når dette compilerflag er aktivt ved begyndelsen af en programblok, kan udlæsninger til skærmen standses med Ctrl–S, og Ctrl–C kan anvendes til at afbryde programmet under ind– og udlæsning. Yderligere detaljer findes i afsnit 17.1

Værdi ved start af compiler: C+

D Kun MS–DOS versionen. Når dette compilerflag er aktivt ved begyndelsen af en programblok, vil der blive foretaget device–check på tekstfiler. Hvis device–check er til og en textfil refererer til en MS–DOS logisk enhed, vil I/O blive foretaget et tegn ad gangen.

Værdi ved start af compiler: D+

F Kun MS–DOS versionen. Denne instruktion skal efterfølges af et tal, der angiver det største antal filer, der kan være åbne på samme tid. Yderligere detaljer findes i afsnit 13.1.

Værdi ved start af compiler: F16

G Kun MS–DOS versionen. Denne instruktion skal efterfølges af et tal, der angiver bufferstørrelsen for "input" standardfilen. Hvis bufferstørrelsen er nul, refererer "input" til CON: enheden. Hvis bufferstørrelsen er forskellig fra nul, refererer "input" til MS–DOS standard input filen. Yderligere detaljer findes i afsnit 13.3.4.

Værdi ved start af compiler: G0

I Et compilerflag, der, når det er aktiveret, instruerer compileren om at generere kode, der checker alle kald til I/O rutiner for at sikre, at der ikke er sket I/O fejl. Yderligere detaljer gives i afsnit 13.5. Hvis I direktivet efterfølges af et filnavn, bliver denne fil medtaget i oversættelsen af programmet. Yderligere detaljer

herom findes i kapitel 18. Bemærk, at et I direktiv med et filnavn ikke kan efterfølges af flere direktiver.

Værdi ved start af compiler: I+

K Kun PolyPascal–86. Når dette compilerflag er aktiveret, genererer compileren kode, der checker for stakoverløb før hver kald til en procedure eller funktion. Yderligere detaljer findes i afsnit 15.8.

Værdi ved start af compiler: K+

P Kun MS–DOS versionen. Denne instruktion skal efterfølges af et tal, der angiver bufferstørrelsen for "output" standardfilen. Hvis bufferstørrelsen er nul, refererer "output" til CON: enheden. Hvis bufferstørrelsen er forskellig fra nul, refererer "output" til MS–DOS standard output filen. Yderligere detaljer findes i afsnit 13.3.4.

Værdi ved start af compiler: P0

R Når dette compilerflag er aktiveret, genererer compileren kode, der checker alle indexeringer af arraystrukturer og alle tilskrivninger til variable af skalare typer og delintervaltyper, for at sikre, at værdierne ligger indenfor de tilladte grænser. Yderligere detaljer gives i afsnit 7.4 og afsnit 9.1

Værdi ved start af compiler: R+

S Kun PolyPascal–80. Et compilerflag, der, når det er aktiveret, instruerer compileren om at optimere array–indexeringer med hensyn til kørselshastighed i stedet for kodestørrelse. Yderligere detaljer findes i afsnit 9.1

Værdi ved start af compiler: S–

U Når dette compilerflag er aktiveret, genererer compileren kode, der udfører et interruptcheck før hver sætning. Når programmet udføres, kan sådanne sætninger afbrydes ved at trykke Ctrl–C på tastaturet. Yderligere detaljer findes i afsnit 17.2.

Værdi ved start af compiler: U–

V Et compilerflag, der, når det er passiveret, instruerer compileren om at tillade strengvariable af enhver type som aktuelle var-parametre, selvom deres maksimale længde ikke stemmer overens med den formelle var-parameter. Yderligere detaljer findes i afsnit 15.6.

Værdi ved start af compiler: V+

W Kun PolyPascal-80. Denne instruktion skal efterfølges af et talciffer (0..9), der angiver det maksimale nestingniveau for WITH sætninger. W direktivet skal anvendes før erklæringsdelen for den blok, det skal gælde for. Yderligere detaljer findes i afsnit 10.2.

Værdi ved start af compiler: W4

Appendix D
ASCII tegntabel

DEC	HX	TEGN	DEC	HX	TEGN	DEC	HX	TEGN	DEC	HX	TEGN
0	00	NUL	32	20	SPACE	64	40	@	96	60	‘
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	”	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	’	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	–	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	Æ	123	7B	æ
28	1C	FS	60	3C	<	92	5C	Ø	124	7C	ø
29	1D	GS	61	3D	=	93	5D	Å	125	7D	å
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Appendix E
PolyPascal syntax

PolyPascal sprogets syntax vises i dette appendix på BNF form (Backus–Naur form). De nedenfor viste symboler indgår i BNF formen, og er ikke en del af Pascal sproget:

::= Betyder 'er defineret som'.

| Betyder 'eller'.

{...} Angiver, at symbolerne mellem parenteserne kan gentages nul eller flere gange.

Symbolet <character> angiver ethvert skrivbart ASCII tegn, dvs. et tegn med en ASCII værdi mellem $20 og $7F.

<empty> ::=

<letter> ::= A | B | C | D | E | F | G | H | I | J | K | L | M |
N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
p | q | r | s | t | u | v | w | x | y | z | _

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<hexdigit> ::= <digit> | A | B | C | D | E | F

<program> ::= <program heading> <block> .

<program heading> ::= <empty> | PROGRAM
<program identifier> <file identifier list>

<program identifier> ::= <identifier>

<identifier> ::= letter { <letter or digit> }

<letter or digit> ::= <letter> | <digit>

<file identifier list> ::= <empty> | (<file identifer>
{ , <file identifer> }

<file identifier> ::= <identifier>

<block> ::= <declaration part> <statement part>

<declaration part> ::= { <declaration section> }

<declaration section> ::= <label declaration part> |
<constant definition part> | <type definition part> |
<variable declaration part> |
<procedure and function declaration part>

<label declaration part> ::= LABEL <label> { , <label> } ;

<label> ::= <letter or digit> { <letter or digit> }

<constant definition part> ::= CONST <constant definition>
{ ; <constant definition> } ;

<constant definition> ::= <untyped constant definition> |
<typed constant definition>

<untyped constant definition> ::= <identifier> = <constant>

<constant> ::= <unsigned number> | <sign> <unsigned number> |
<constant identifier> | <sign> <constant identifier> |
<string>

<unsigned number> ::= <unsigned integer> | <unsigned real>

<unsigned integer> ::= <digit sequence> | $ <hexdigit sequence>

<digit sequence> ::= <digit> { <digit> }

<hexdigit sequence> ::= <hexdigit> { <hexdigit> }

<unsigned real> ::= <digit sequence> . <digit sequence> |
<digit sequence> . <digit sequence> E <scale factor> |
<digit sequence> E <scale factor>

<scale factor> ::= <digit sequence> | <sign> <digit sequence>

<sign> ::= + | -

<constant identifier> ::= <identifier>

<string> ::= { <string element> }

<string element> ::= <text string> | <control character>

<text string> ::= ' { <character> } '

<control character> ::= @ <unsigned integer> | ^ <character>

<structured constant definition> ::= <identifier> : <type> =
    <structured constant>

<type> ::= <simple type> | <structured type> | <pointer type>

<simple type> ::= <scalar type> | <subrange type> |
    <type identifier>

<scalar type> ::= (<identifier> { , <identifier> })

<subrange type> ::= <constant> .. <constant>

<type identifier> ::= <identifier>

<structured type> ::= <unpacked structured type> |
    PACKED <unpacked structured type>

<unpacked structured type> ::= <string type> | <array type> |
    <record type> | <set type> | <file type>

<string type> ::= STRING [<constant>]

<array type> ::= ARRAY [<index type> { , <index type> }] OF
    <component type>

<index type> ::= <simple type>

<component type> ::= <type>

<record type> ::= RECORD <field list> END

<field list> ::= <fixed part> | <fixed part> ; <variant part> |
<variant part>

<fixed part> ::= <record section> { ; <record section> }

<record section> ::= <empty> | <field identifier>
{ , <field identifier> } : <type>

<field identifier> ::= <identifier>

<variant part> ::= CASE <tag field> <type identifier> OF
<variant> { ; <variant> }

<tag field> ::= <empty> | <field identifier> :

<variant> ::= <empty> | <case label list> : (<field list>)

<case label list> ::= <case label> { , <case label> }

<case label> ::= <constant>

<set type> ::= SET OF <base type>

<base type> ::= <simple type>

<file type> ::= FILE OF <type>

<pointer type> ::= ^ <type identifier>

<structured constant> ::= <constant> | <array constant> |
<record constant> | <set constant>

<array constant> ::= (<structured constant>
{ , <structured constant> })

<record constant> ::= (<record constant element>
{ ; <record constant element> })

<record constant element> ::= <field identifier> :
<structured constant>

<set constant> ::= [{ <set constant element> }]

<set constant element> ::= <constant> | <constant> .. <constant>

<type definition part> ::= TYPE <type definition>
{ ; <type definition> } ;

<type definition> ::= <identifier> = <type>

<variable declaration part> ::= VAR <variable declaration>
{ ; <variable declaration> } ;

<variable declaration> ::= <identifier list> : <type> |
<identifier> : <type> AT <address specification>

<identifier list> ::= <identifier> { , <identifier> }

<address specification> ::= <variable identifier> | <constant> |
<constant> : <constant> | DSEG : <constant> |
CSEG : <constant>

<procedure and function declaration part> ::=
{ <procedure or function declaration> }

<procedure or function declaration> ::= <procedure declaration> |
<function declaration>

<procedure declaration> ::= <procedure heading> <block> ; |
OVERLAY <procedure heading> <block> ;

<procedure heading> ::= PROCEDURE <identifier> ; |
PROCEDURE <identifier> (<formal parameter section>
{ , <formal parameter section> }) ;

<formal parameter section> ::= <parameter group> |
VAR <parameter group>

<parameter group> ::= <identifier list> : <type identifier>

<function declaration> ::= <function heading> <block> ; |
OVERLAY <function heading> <block> ;

<function heading> ::= FUNCTION <identifier> : <result type> ; |
FUNCTION <identifier> (<formal parameter section>
{ , <formal parameter section> }) : <result type> ;

<result type> ::= <type identifier>

<statement part> ::= <compound statement>

<compound statement> ::= BEGIN <statement>
{ ; <statement> } END

<statement> ::= <simple statement> | <structured statement>

<simple statement> ::= <assignment statement> |
<procedure statement> | <goto statement> |
<code statement> | <empty statement>

<assignment statement> ::= <variable> := <expression> |
<function identifier> ::= <expression>

<variable> ::= <entire variable> | <component variable> |
<referenced variable>

<entire variable> ::= <variable identifier> |
<typed constant identifier>

<variable identifier> ::= <identifier>

<typed constant identifier> ::= <identifier>

<component variable> ::= <indexed variable> | <field designator>

<indexed variable> ::= <array variable> [<expression>
{ , <expression> }]

<array variable> ::= <variable>

<field designator> ::= <record variable> . <field identifier>

<record variable> ::= <variable>

<field identifier> ::= <identifier>

<referenced variable> ::= <pointer variable> ^

<pointer variable> ::= <variable>

<expression> ::= <simple expression> { <relational operator>
<simple expression> }

<simple expression> ::= <term> { <adding operator> <term> }

<term> ::= <complemented factor> { <multiplying operator>
<complemented factor> }

<complemented factor> ::= <signed factor> | NOT <signed factor>

<signed factor> ::= <factor> | <sign> <factor>

<factor> ::= <variable> | <unsigned constant> |
(<expression>) | <function designator> | <set>

<unsigned constant> ::= <unsigned number> | <string> |
<constant identifier> | NIL

<function designator> ::= <function identifier> |
<function identifer> (<actual parameter>
{ , <actual parameter> })

<function identifier> ::= <identifer>

<actual parameter> ::= <expression> | <variable>

<set> ::= [{ <set element> }]

<set element> ::= <expression> | <expression> .. <expression>

<multiplying operator> ::= * | / | DIV | MOD | AND | SHL | SHR

<adding operator> ::= + | - | OR | EXOR

<relational operator> ::= = | <> | >= | <= | > | < | IN

<procedure statement> ::= <procedure identifier> |
<procedure identifier> (<actual parameter>
{ , <actual parameter> })

<goto statement> ::= GOTO <label>

<code statement> ::= CODE <code element> { , <code element> }

```
<code element> ::= <unsized code element> |
      < <unsized code element> | > <unsized code element>

<unsized code element> ::= <data element>
      { <sign> <data element> }

<data element> ::= <integer constant> | <variable identifier> |
      <procedure identifier> | <function identifier> | *

<integer constant> ::= <constant>

<empty statement> ::= <empty>

<structured statement> ::= <compound statement> |
      <conditional statement> | <repetitive statement> |
      <with statement>

<conditional statement> ::= <if statement> | <case statement>

<if statement> ::= IF <expression> THEN <statement> |
      IF <expression> THEN <statement> ELSE <statement>

<case statement> ::= CASE <expression> OF <case element>
      { ; <case element> } END | CASE <expression> OF
      <case element> { ; <case element> } OTHERWISE
      <statement> { ; <statement> } END

<case element> ::= <case list> : <statement>

<case list> ::= <case list element> { , <case list element> }

<case list element> ::= <constant> | <constant> .. <constant>

<repetitive statement> ::= <while statement> |
      <repeat statement> | <for statement>

<while statement> ::= WHILE <expression> DO <statement>

<repeat statement> ::= REPEAT <statement> { ; <statement> }
      UNTIL <expression>

<for statement> ::= FOR <control variable> := <for list> DO
      <statement>
```

<for list> ::= <initial value> TO <final value> |
<initial value> DOWNTO <final value>

<control variable> ::= <variable identifier>

<initial value> ::= <expression>

<final value> ::= <expression>

<with statement> ::= WITH <record variable list> DO
<statement>

<record variable list> ::= <record variable>
{ , <record variable> }

Appendix F
I/O fejl

En I/O fejl rapporteres når der opstår en fejl under en filoperation eller en ind- eller udlæsningsoperation. Hvis sætningen hvori fejlen opstår er oversat i {$I+} stillingen, stopper programmet, og en fejlmeddelelse vises:

```
I/O ERROR nn AT PC=aaaa
Program terminated
```

hvor nn er fejlens nummer (i hexnotation), og aaaa er fejlens relative adresse (i forhold til startadressen på programkoden). I {$I-} stillingen stoppes programmet ikke, men i stedet gemmes fejlens nummer i en systemvariabel, der kan undersøges ved at kalde iores funktionen.

De følgende I/O fejl kan forekomme:

01 Uoverensstemmende elementlængder (kun CP/M versionerne). Denne fejl rapporteres af reset hvis programmet prøver på at sammenknytte en typeangiven filvariabel (FILE OF type) med en diskfil, der ikke har den samme elementlængde (og dermed ikke den samme elementtype).

02 Filen findes ikke. Denne fejl rapporteres af append, chain, chdir, erase, execute, mkdir, rename, reset eller rmdir hvis der ikke findes en fil af det navn, der er tilknyttet filvariablen.

03 Biblioteket er fuldt. Denne fejl rapporteres af rewrite, hvis der ikke er plads til nye filer i diskens bibliotek. MS-DOS versionen kan også rapportere fejl 03, hvis filnavnet er forkert opbygget.

04 Fil forsvundet. Denne fejl rapporteres af close hvis den diskfil, der refereres til, er forsvundet. Fejlen kan eventuelt skyldes at erase er blevet kaldt, eller at brugeren har isat en anden diskette.

05 Indlæsning ej tilladt. Denne fejl rapporteres af read eller readln hvis programmet prøver at læse fra en textfil, der ikke er forberedt med reset proceduren.

06 Udlæsning ej tilladt. Denne fejl rapporteres af write eller writeln hvis programmet prøver at skrive til en textfil, der ikke er forberedt med rewrite eller append proceduren.

07 For mange åbne filer (kun MS–DOS versionen). Denne fejl rapporteres af append, chain, execute, reset eller rewrite, hvis den interne filtabel i PolyPascal eller MS–DOS er fuld. Se afsnit 13.1 for yderligere detaljer.

08 Denne fejlkode er ubrugt.

09 Fejl i numerisk værdi. Denne fejl rapporteres af read eller readln, fra en textfil, hvis en indlæst numerisk værdi ikke er af et korrekt format.

0A Denne fejlkode er ubrugt.

0B Fil–overløb (kun CP/M versionerne). Denne fejl rapporteres af write (til en typeangiven fil) hvis programmet prøver at lagre mere end 65535 elementer i en fil.

0C Læsefejl. Denne fejl rapporteres af read ved læsning (fra en typeangiven fil) ud over slutningen af filen.

0D Skrivefejl. Denne fejl rapporteres af write, writeln eller flush hvis filen ikke kan udvides yderligere fordi disken er fuld.

0E Ulovligt elementnummer. Denne fejl rapporteres af seek hvis programmet prøver på at flytte fil–pointeren til et element, der ligger ud over filens længde.

0F Fil ej åben. Denne fejl rapporteres af read, blockread, write eller blockwrite, hvis filen ikke er åben.

10 Logisk I/O enhed tillades ikke her. Denne fejl rapporteres af erase, rename, execute eller chain (på en textfil) hvis filen er tilskrevet en logisk I/O enhed.

11 Ulovlig brug af programkædning. Denne fejl rapporteres af execute eller chain hvis et program, der blev startet med RUN kommandoen, prøver på at starte et andet program. execute og chain må kun anvendes fra programfiler.

12 Ulovlig brug af assign. Denne fejl rapporteres af assign hvis et program prøver på at tilskrive et filnavn til en af standardfilerne (input, output, con, trm, kbd, lst, aux eller usr).

Appendix G
Kørselsfejl

En kørselsfejl rapporteres, når der opstår en uoprettelig fejltilstand i systemet. Kørselsfejl betyder altid, at programmet stopper og udskriver en fejlmeddelelse:

```
EXECUTION ERROR nn AT PC=aaaa
Program terminated
```

hvor nn er fejlens nummer (i hexnotation), og aaaa er fejlens relative adresse (i forhold til programmets startadresse).

De følgende kørselsfejl kan forekomme:

01 Streng–overløb. Denne fejl rapporteres ved sammensætning af af to strenge (med plus operatoren eller concat proceduren), hvis den resulterende streng er længere end 255 tegn, eller ved konvertering af en streng til et tegn, hvis strengens længde er forskellig fra 1.

02 Ulovlig strengposition. Denne fejl rapporteres af copy, delete eller insert, hvis strengpositionen ikke er en værdi mellem 1 og 255.

03 Overløb under floating point operation.

04 Division med nul forsøgt.

05 Ulovligt argument til sqrt. Programmet forsøger at udregne kvadratroden til et negativt tal.

06 Ulovligt argument til ln. Programmet forsøger at udregne den naturlige logaritme til et tal, der er mindre end eller lig med 0.

07 Ulovligt argument til trunc eller round. Det reelle tal, der forsøges konverteret til et heltal, er ikke indenfor heltalsområdet (–32768..32767).

08 Over/underløb på index. Værdien af et indexudtryk er udenfor de tilladte grænser.

09 Over/underløb på skalar eller delinterval. Værdien, der forsøges tilskrevet til en variabel af en skalar type eller en delintervaltype, er udenfor de tilladte grænser.

0A Kollision mellem heap og stak. Denne fejl rapporteres ved et kald til new, eller ved et kald til et rekursivt underprogram, hvis der ikke er tilstrækkelig "luft" (frit lager) mellem heap–pointeren og procedurestakpointeren (kontrolleres kun i {$K+} tilstanden).

0B Overlay fil ikke fundet. Fejlen gives ved indgangen til overlay subrutinen, hvis den aktuelle overlay fil ikke findes, eller hvis den er forkert (ødelagt).

Appendix H
Compilerfejl

01 '.' forventet.

02 BEGIN forventet.

03 Ulovlig resultattype. Funktioner kan kun returnere værdier af skalare typer, strengtyper og pointertyper.

04 Dubleret identifier. Dette navn er allerede anvendt.

05 Absolutte variable tillades ikke i poster. AT specifikationen kan ikke anvendes inden i poster.

06 Typeidentifier forventet.

07 Filer må kun være VAR parametre.

08 Fejlagtig eller ukendt type.

09 END forventet.

10 Mængdes grundtype er for stor. Grundtypen i en mængde skal være en skalar eller et delinterval, der ikke har over 256 mulige værdier, og begge grænser skal være mellem 0 og 255.

11 Filelementer må ikke være filer.

12 Ulovlig strenglængde. En strengs maksimumlængde skal være et heltal mellem 1 og 255.

13 Ulovlig grundtype for delinterval. Alle skalare typer, undtagen reals, er tilladte.

14 '..' forventet.

15 Uens typer i intervalgrænser. Typen af den nedre grænse stemmer ikke overens med typen af den øvre.

16 Øvre grænse større end nedre. Den ordinale værdi af den øvre grænse skal være større end eller lig den ordinale værdi af den nedre.

17 Fejlagtig eller ukendt simpel type.

18 Simpel type forventet. Alle skalare typer, undtagen reals, er simple typer.

19 Ukendt pointertype i typedefinition(er). En af de ovenstående typedefinitioner refererer til en ukendt typeidentifier.

20 Udefineret label i sætningsdel. Den ovenstående sætningsdel indeholder en reference til en ukendt label.

21 Ulovlig GOTO i sætningsdel. En GOTO sætning i den ovenstående sætningsdel refererer til en label inden i en FOR løkke.

22 Label er allerede defineret.

23 THEN forventet.

24 DO forventet.

25 Fejlagtig eller ukendt variabelidentifier.

26 Variabeltype er ikke en simpel type. En FOR sætnings kontrolvariabel skal være af en simpel type.

27 Uens typer i FOR sætnings udtryk. Typen af et (eller begge) af en FOR sætnings kontroludtryk stemmer ikke overens med kontrolvariablens type.

28 TO eller DOWNTO forventet.

29 Konstant er ikke af samme type som CASE udtryk.

30 END eller OTHERWISE forventet.

31 Ukendt label.

32 For mange nestede WITH sætninger. Brug compilerens W direktiv til at forøge det maksimale antal nestede WITH sætninger.

33 Postvariabel forventet.

34 Fejlagtig eller ukendt variabel.

35 Ulovlig tilskrivning. Tilskrivning til filer og typeløse parametre tillades ikke.

36 Uens typer i tilskrivning eller parameterliste. Typen af variablen og udtrykket i en tilskrivning stemmer ikke overens, eller typen af den aktuelle parameter i et kald til et underprogram stemmer ikke overens med typen af den formelle parameter.

37 Udtryk er ikke af typen integer.

38 Udtryk er ikke af typen boolean.

39 Udtryk er ikke af en simpel type. Alle skalare typer, undtagen real, er simple typer.

40 Udtryk er ikke af en strengtype.

41 Uens typer i udtryk. Typerne af operanderne i udtrykket er ikke kompatible.

42 Operandtype(r) stemmer ikke overens med operator.

43 Strukturerede variable tillades ikke her. Arraystrukturer (undtagen tegn–arrays), poster og filer tillades ikke her.

44 Uens typer i mængde. Typen af elementerne eller intervallerne i en mængde stemmer ikke overens.

45 Syntaksfejl eller ukendt identifier i udtryk.

46 Konstanter tillades ikke her.

47 Udtrykstype er forskellig fra indextype.

48 Fejlagtig eller ukendt feltidentifier.

49 Fejlagtig eller ukendt konstant.

50 Integer konstant forventet.

51 Integer eller real konstant forventet.

52 Strengkonstant ikke afsluttet. Strengkonstanter må ikke strække sig over flere linier.

53 Fejl i integer konstant. Enten er der en syntaksfejl i konstanten, eller også er den udenfor heltalsområdet (–32768 til 32767). Bemærk, at hele reelle tal skal efterfølges af et decimalpunktum og et nul, f.eks. 14765123.0. Formatet af heltalskonstanter er defineret i afsnit 2.2.

54 Fejl i real konstant. Formatet af en real konstant er defineret i afsnit 2.2.

55 Ulovligt tegn i identifier. Det første tegn i en identifier skal være et bogstav ('A' til 'Z' og 'a' til 'z') eller en underscore ('_'). De følgende tegn skal være bogstaver, talcifre ('0' til '9') eller underscores.

56 '[' eller '(.' forventet.

57 ']' eller '.)' forventet.

58 ':' forventet.

59 ';' forventet.

60 Syntaksfejl eller ukendt identifier i sætning.

61 ',' forventet.

62 '(' forventet.

63 ')' forventet.

64 '=' forventet.

65 ':=' forventet.

66 OF forventet.

67 Programmet kan ikke slutte her. Denne fejl rapporteres, hvis compileren ikke regner programmet for afsluttet ved slutningen af programteksten. Muligvis er der flere BEGIN'er, end der er END'er.

68 Fil findes ikke. Den angivne "include–fil" findes ikke.

69 Buffer overflow. Denne fejl rapporteres, hvis der ikke er nok lager til at oversætte programmet i. Bemærk, at fejlen kan forekomme, selvom der tilsyneladende er lager tilovers – dette lager bruges imidlertid til oversætterens symboltabel. Del teksten ind i mindre afsnit, og brug "include–filer".

70 Memory overflow. Der er ikke plads til programmet og dets variable i lageret.

71 Variable af denne type kan ikke indlæses.

72 Variable af denne type kan ikke udlæses.

73 Tekstfil forventet.

74 Filvariabel forventet.

75 Tekstfiler tillades ikke her.

76 Typeløse filer tillades ikke her.

77 Strengkonstant forventet.

78 Strenglængde stemmer ikke overens med type.

79 Fejlagtig rækkefølge af feltkonstanter.

80 Uens typer i struktureret konstant.

81 Konstant udenfor tilladte grænser.

82 Filer og pointere tillades ikke her.

83 Fejlagtig brug af retype–facilitet. Retype–faciliteten tillader kun simple typer, altså alle skalarer undtagen reals.

84 Integer eller real udtryk forventet.

85 Strengvariabel forventet.

86 Textfiler og typeløse filer tillades ikke her.

87 Typeløs fil forventet.

88 Pointervariabel forventet.

89 Integer eller real variabel forventet.

90 Integer variabel forventet.

91 Reserveret ord.

92 Label er ikke indenfor denne blok. GOTO sætninger må ikke hoppe ud af den nuværende blok.

93 Procedure eller funktion er ikke defineret korrekt. Et underprogram i den ovenstående erklæringsdel er FORWARD specificeret, men programblokken er ikke defineret senere.

94 Fejl i CODE sætning. CODE sætninger beskrives i kapitel 20.

95 Ulovlig brug af AT specifikation. Der kan kun erklæres en variabel ad gangen med AT specifikationen.

96 Overlays tillades ikke i direct mode. Overlays kan ikke anvendes i programmer, der oversættes med en COMPILE eller RUN kommando.

97 Overlays kan ikke FORWARD specificeres.

98 PROCEDURE eller FUNCTION forventet.

99 Overlayfil kan ikke åbnes. Compileren kan ikke åbne en ny overlayfil. Denne fejl kan eventuelt skyldes, at diskens bibliotek er fuldt.

100 Fejl i EXTERNAL fil. Denne fejl rapporteres af CP/M–86 versionen af PolyPascal–86 hvis en EXTERNAL fil ikke overholder CP/M systemets 'CMD' filformat.

101 Nestede "include–filer" tillades ikke.

102 Fejl i compilerdirektiv.

Index

A

A compilerdirektiv, 112
abs, 106
Absolut adresserede variable, 24
Absolutte procedurer og funktioner, 112
Adderende operatorer, 28
addr, 97, 108
Afbrydning af programmer, 127
Aktuelle parametre, 99
allocate, 95, 97
alloff, 49
AND operatoren, 28
append, 79
arctan, 106
Aritmetiske funktioner, 106
Arrayerklæringer, 51
Arrayindexering, 51
Arraykonstanter, 68
Arrayoptimering, 52
Arrays,
  flerdimensionale, 52
  format af, 154
  mem, 53
  port, 54
  prædefinerede, 53
  tegn, 55
Arraystrukturer, 51
Arraytilskrivninger, 53
ASCII tegntabel, 189
assign, 73
AT specifikation, 24
aux, 84
AUX: enheden, 81, 143

B

B compilerdirektiv, 123, 124
bdos, 140
bdosb, 141
BEGIN/END blokke, 33
Betingede sætninger, 33
bios, 140
biosb, 141
blkoff, 49
blkon, 49
blockread, 87
blockwrite, 87
BNF syntax, 191
boolean, 20
Brugerdefinerede I/O drivere, 143
buflen, 123
byte, 20

C

C compilerdirektiv, 127
CASE i poster, 60
CASE sætninger, 34
chain, 131
char, 20
chdir, 74
chr, 107
close, 74
clreol, 49
clreos, 49
clrhom, 49
CODE sætninger, 135
Compilerdirektiver, 17, 185
  A, 112
  B, 123, 124
  C, 127
  F, 72
  G, 85
  I, 89, 129
  K, 113
  P, 85
  R, 40, 48, 51
  S, 52
  U, 127

V, 110
W, 60
Compilerfejl, 207
con, 84
CON: enheden, 81, 122, 143
concat, 43
copy, 43
cos, 106
cseg, 109
Ctrl-C, 127
Ctrl-S, 127

D

Danske tegn, 11
Dataformater, 147
Datasegment, 163
delete, 43
Delintervaller, 38
dellin, 49
Diskfiler, format af, 155
DIV operatoren, 28
DO,
i FOR sætninger, 36
i WHILE sætninger, 35
DOWNTO, i FOR sætninger, 36
dseg, 109

E

Editeringsnøgler, 122
ELSE, i IF sætninger, 33
end-of-file mærke, 77, 83
end-of-line mærke, 77, 83
eof, 75, 79
textfiler, 83
eoln, 79, 83
erase, 74
Erklæringsdel, 21
ERR: enheden, 82
execute, 131
Execution error, 205
exit, 103
EXOR operatoren, 28
exp, 106
EXTERNAL specifikationer, 118
EXTERNAL subrutiner,
parameteroverførsel, 156, 158

F

F compilerdirektiv, 72
Fejl,
compiler, 207
I/O, 201
kørsel, 205
Fil interface blok, format af, 150
Filbehandlingsfunktioner, 75
Filbehandlingsprocedurer, 73
Filer, 71
erklæring, 71
operationer på, 73
standard, 83
text, 77
typeløse, 87
Filkopieringsprogram, 88
fill, 103
Filnavne, 72
Flerdimensionale arrays, 52
flush, 74
textfiler, 79
fmt, 45
FOR sætninger, 36
Formater, interne data, 147
FORWARD specifikationer, 109
frac, 107
Funktioner, 99, 104
absolutte, 112
Funktioner som parametre, 176
Funktionserklæringer, 25, 104
Funktionskald, udtryk, 29
Funktionsoverskrift, 104
Funktionsresultater, format
af, 157, 160
Fælles variable, 132

G

G compilerdirektiv, 85
gcdir, 75
get, 175
GOTO sætninger, 32
gotoxy, 103
Grundlæggende symboler, 11

H

halt, 103
heap, 95, 166
Heap-pointeren, 164
Heappointer, 93
Heapsegment, 164
hi, 108
hptr, 164, 167

I

I compilerdirektiv, 89, 129
I/O dataporte, 54
I/O drivere, 143
I/O fejl, 201
I/O kontrol, 89
I/O omdirigering, 84
I/O redirection, 84
Identifiere, 15
  standard, 12
IF sætninger, 33
IN operator, 65
In-line maskinkode, 135
Include-filer, 129
Indexering, 51
Indexkontrol, 51
Indlæsning og udlæsning, 121
INP: enheden, 81
input, 84
Input/output, 121
insert, 43
inslin, 49
int, 106
integer, 19
Integer overløb, 19
Interne dataformater, 147
Interruptprocedurer, 169
  PolyPascal-80, 171
  PolyPascal-86, 169
Interrupts, hardware, 169
  software, 139
intoff, 49
inton, 49
iores, 89

K

K compilerdirektiv, 113
kbd, 84
KBD: enheden, 81, 143
keypress, 108
Kodesegment, 163
Kommentarer, 16
Konstantdefinitioner, 22
Konstanter,
  array, 68
  mængder, 70
  numeriske, 15
  poster, 69
  prædefinerede, 22
  streng, 16
  strukturerede, 68
  tegn, 16
  typeangivne, 67
Kontroltegn, 16
Konverteringsfunktioner, 107
Kædning af programmer, 131
Kørselsfejl, 205

L

Labelerklæringer, 22
Labels, rækkevidde, 32
Lagerorganisering, 163
  PolyPascal-80, 164
  PolyPascal-86, 163
len, 42
length, 75
ln, 106
lo, 108
Logiske enheder, 80
  brugerdefinerede drivere, 143
longlen, 75
longpos, 75
lst, 84
LST: enheden, 81, 143

M

mark, 94, 97
Maskinkode, 135
mem, 53
memavail, 95, 97

memw, 54
mkdir, 75
MOD operatoren, 28
move, 103
Multiplicerende operatorer, 28
Mængde, tom, 65
Mængdeangivere, 64
Mængdekonstanter, 70
Mængder, 63
  erklæring, 64
  format af, 149
Mængderoperatorer, 65
Mængdetilskrivninger, 66
Mængdeudtryk, 64
Mærkefelt, 61

N

Negationsoperatoren, 27
new, 92, 96
NIL pointer, 92
NOT operatoren, 27
Numeriske konstanter, 15

O

odd, 107
ofs, 109
Omdirigering af I/O, 84
Operatorer, 27, 183
  mængder, 65
OR operatoren, 28
ord, 38, 97, 107
OTHERWISE, i CASE sætninger, 34
OUT: enheden, 81
output, 84
Output/input, 121
ovdrive, 118
Overlay procedurer og funktioner, 113
Oversigt over compiler-direktiver, 185
Oversigt over operatorer, 183
Oversigt over standard-funktioner, 177
Oversigt over standard-procedurer, 177
ovpath, 117

P

P compilerdirektiv, 85
PACKED, 176
page, 176
Pakkede variable, 176
Parametre, 99
  format af, 156, 158
  strenge, 110
  typeløse, 111
Pointer rutiner, 96
Pointere, 91
  direkte adgang til, 95
  format af, 96, 154
  kædede lister, 93
Pointererklæringer, 92
Pointertilskrivninger, 92
Pointerudtryk, 92
port, 54
pos, 43
position, 75
Poster, 57
  erklæring, 57
  felter, 58
  format af, 154
  mærkefelt, 61
  tilskrivning, 58
  varianter i, 60
Postkonstanter, 69
pred, 38, 107
Procedureerklæringer, 25, 100
Procedureoverskrift, 100
Procedurer, 99, 100
  absolutte, 112
Procedurer som parametre, 176
Proceduresætninger, 32
Programafbrydning, 127
Programkædning, 131
Programlinier, 14
Programoverskrift, 21
Prædefinerede arraystrukturer, 53
Prædefinerede strenge, 49
ptr, 97
put, 175
pwrten, 107

Q

Quicksort, 85

R

R compilerdirektiv, 40, 48, 51
random, 108
randomize, 103
read, 73, 121
  char, 121
  fra CON: enheden, 122
  numerisk variabel, 122
  streng, 121
readln, 79, 123
Reals, format af, 148
Records (se poster), 57
Redirection af I/O, 84
regpack, 139
Rekursionsstak, 166
Rekursive funktioner, 104, 105
Rekursive procedurer, 101
Relationelle operatorer, 28
release, 94, 97
rename, 74
REPEAT sætninger, 35
Repetitionssætninger, 35
Reserverede ord, 12
reset, 73
  typeløse filer, 87
rewrite, 73
  typeløse filer, 87
rmdir, 75
round, 107
rptr, 167
Run-time error, 205
rvsoff, 49
rvson, 49

S

S compilerdirektiv, 52
Sammensatte sætninger, 33
seek, 74
seg, 109
Segmenter, 163
seof, 79
seoln, 79
Separatorer, 14
SHL og SHR operatorerne, 28
Simple sætninger, 31
sin, 106
Skalare funktioner, 107
Skalare typer, 37
  standard, 19
Skalarer, format af, 148
Software interrupts, 139
sptr, 167
sqr, 106
sqrt, 106
sseg, 109
Stak, 166
Stak overflow kontrol, 113, 167
Staksegment, 164
Standard identifiere, 12
Standard Pascal, forskelle, 175
Standard typer, 19
Standardfiler, 83
Standardfunktioner, 106, 177
  abs, 106
  addr, 97, 108
  arctan, 106
  bdos, 140
  bdosb, 141
  bios, 140
  biosb, 141
  chr, 107
  concat, 43
  copy, 43
  cos, 106
  cseg, 109
  dseg, 109
  eof, 75
  exp, 106
  fmt, 45
  frac, 107
  hi, 108
  int, 106
  iores, 89
  keypress, 108
  len, 42
  length, 75
  ln, 106
  lo, 108
  longlen, 75

longpos, 75
memavail, 97
odd, 107
ofs, 109
ord, 97, 107
pos, 43
position, 75
pred, 107
ptr, 97
pwrten, 107
random, 108
round, 107
seg, 109
sin, 106
size, 108
sqr, 106
sqrt, 106
sseg, 109
succ, 107
swap, 108
trunc, 107
Standardprocedurer, 103, 177
allocate, 97
append, 79
assign, 73
bdos, 140
bios, 140
blockread, 87
blockwrite, 87
chain, 131
chdir, 74
close, 74
delete, 43
eof, 79
eoln, 79
erase, 74
execute, 131
exit, 103
fill, 103
flush, 74
gcdir, 75
gotoxy, 103
halt, 103
insert, 43
mark, 97
mkdir, 75
move, 103
new, 96
ovdrive, 118
ovpath, 117
randomize, 103
read, 73, 121
readln, 79, 123
release, 97
rename, 74
reset, 73
rewrite, 73
rmdir, 75
seek, 74
seof, 79
seoln, 79
str, 44
swint, 139
truncate, 74
val, 44
write, 73, 124
writeln, 79, 126
str, 44
Strenge,
format af, 149
prædefinerede, 49
Strenge som parametre, 110
Strengformatering, 45
Strengfunktioner, 42
Strengindexering, 48
Strengkonstanter, 16
Strengprocedurer, 43
Strengtilskrivninger, 42
Strengtyper, 41
definering af, 41
Strengudtryk, 41
Strukturerede konstanter, 68
Strukturerede sætninger, 32
succ, 38, 107
swap, 108
swint, 139
Syntax beskrivelse, 191
Systemkald, 139
Sætninger, 31
Sætningsdel, 25

T

Tag field, 61
Tegn, 20
  ASCII tabel, 189
Tegn-arrays, 55
Tegnkonstanter, 16
Textfiler, 77
  erklæring, 78
  format af, 155
  operationer på, 78
Tilskrivninger,
  arrays, 53
  mængder, 66
  pointere, 92
  poster, 58
Tilskrivningssætninger, 31
TO, i FOR sætninger, 36
Tom mængde, 65
Tomme sætninger, 32
TRM: enheden, 81
trunc, 107
truncate, 74
Typeangivne filer, format af, 155
Typeangivne konstanter, 67
  simple, 67
Typedefinitioner, 23
Typekonvertering, 39
Typeløse filer, 87
  erklæring, 87
  operationer på, 87
Typeløse parametre, 111
Typer, standard, 19

U

U compilerdirektiv, 127
Udtryk, 27
  mængder, 64
  pointer, 92
  strenge, 41
ulnoff, 49
ulnon, 49
UNTIL, i REPEAT sætninger, 35
usr, 84
USR: enheden, 81, 143

V

V compilerdirektiv, 110
val, 44
Value-parametre, 99, 101
Var-parametre, 99, 102
Variabelerklæringer, 23
Variable,
  allokering af, 147
  faste adresser, 24
  format af, 147
Varianter, 60
Værdikontrol,
  skalarer, 40
  strenge, 48

W

W compilerdirektiv, 60
WHILE sætninger, 35
WITH sætninger, 59
write, 73, 124
  formateret, 125
  logisk variabel, 124
  numerisk, 124
  streng, 124
  tegn, 124
writeln, 79, 126

PolyFile

PolyPascal File Access System

Version 1.1

REFERENCE MANUAL

PolyData MicroCenter A/S
Åboulevarden 13
DK–1960 København V

PolyData MicroCenter A/S
Åboulevarden 13
DK–1960 København V

Telefon: (01) 35 61 66
Telex: 27439 poly dk

Indholdsfortegnelse

0 Indledning 5

1 PolyFile oversigt 7

1.1 Bestanddele 7
1.2 Funktionsoversigt 7
1.2.1 Datafil klargøring og vedligeholdelse 8
1.2.2 Indexfil klargøring 8
1.2.3 Indexfil vedligeholdelse 8
1.2.4 Indexfil søgning 9
1.3 PolyFile's typer og variable 9
1.4 Generelt om PolyFile 9
1.5 Brug af PolyFile 10
1.6 Programstruktur med PolyFile 13

2 PolyFile rutiner 15

2.1 Datafil vedligeholdelsesrutiner 15
2.1.1 makefile rutinen 16
2.1.2 openfile rutinen 16
2.1.3 closefile rutinen 17
2.1.4 addrec rutinen 17
2.1.5 deleterec rutinen 18
2.1.6 getrec rutinen 18
2.1.7 putrec rutinen 18
2.1.8 filelen functionen 19
2.1.9 usedrecs functionen 19
2.2 Indexfil vedligeholdelsesrutiner 19
2.2.1 initindex rutinen 21
2.2.2 makeindex rutinen 21
2.2.3 openindex rutinen 22
2.2.4 closeindex rutinen 22
2.2.5 addkey rutinen 23
2.2.6 deletekey rutinen 23
2.2.7 findkey rutinen 24
2.2.8 searchkey rutinen 25
2.2.9 nextkey rutinen 25
2.2.10 prevkey rutinen 26
2.2.11 clearkey rutinen 27

3 Programeksempler **29**

4 PolyFile fejlhåndtering **35**

Index **39**

Indledning

PolyFile er et komplet sæt hjælperutiner til opbygning og vedligeholdelse af datafiler og indexfiler. Rutinerne, der alle er skrevet i PolyPascal, optager mellem 4K og 9K bytes kode.

PolyFile's datafiler indeholder af fast længde dataposter på samme måde som PolyPascal's typeangivne filer (FILE OF type). Posterne i en datafil skal være på mindst 8 bytes, men der er ingen øvre grænse. PolyFile genbruger automatisk slettede poster, før en datafil udvides.

Indexfiler indeholder nøgleværdier (strenge) og disses tilhørende postnumre i datafilen. PolyFile ordner indexfiler efter en højdebalanceret flervejs træstruktur (et B–træ), hvilket sikrer et minimum af disklæsninger til gennemsøgning af et index. For eksempel behøves der kun tre disklæsninger for at gennemsøge en indexfil med 50000 nøgler, hvis der er op til 32 nøgler pr. knude.

Indexfilernes poster (knuder) gemmes i buffere ved hjælp af en "sidst–brugt" prioriteringsalgoritme, hvilket yderligere reducerer tilgang til disken: Hvergang en indexpost findes i en buffer, spares en disktilgang. Indexfilbufferne deles af alle indexfiler, hvorfor der kun skal reserveres et bufferareal.

For en komplet beskrivelse af B–Træ strukturer kan vi anbefale følgende bøger:

> Wirth, Niklaus: Algorithms + Data Structures = Programs. Prentice–Hall, 1976.
>
> Knuth, D.E.: The Art Of Computer Programming, Volume 3:
> Sorting and Searching. Addison–Wesley Publishing, 1973.

PolyFile systemet og den tilhørende dokumentation er forfattet af Anders Hejlsberg og Christen Fihl.

1. PolyFile oversigt

1.1 Bestanddele

PolyFile består af fire PolyPascal kildetekstfiler. PolyFile modulerne inkluderes i et program ved hjælp include–file compilerdirektivet {$I filnavn}. PolyFile filerne er:

DATMAN.PAS Indeholder de grundlæggende data/index fil opstart– og vedligeholdesesrutiner. Dette modul skal altid inkluderes før andre PolyFile moduler.

NPFKEY.PAS Indeholder nextkey, prevkey, findkey og searchkey rutinerne. Disse rutiner udgør tilsammen søgefunktionerne i PolyFile.

ADDKEY.PAS Indeholder addkey rutinen, der bruges til at oprette nye nøgler i indexfiler.

DELKEY.PAS Indeholder deletekey rutinen, der bruges til at slette nøgler i indexfiler.

DATMAN modulet skal altid inkluderes i et program, der bruger PolyFile, og det skal altid være det første modul. NPFKEY, ADDKEY og DELKEY kan efter behov inkluderes i vilkårlig rækkefølge efter DATMAN.

1.2 Funktionsoversigt

PolyFile's funktioner kan grupperes i fire kategorier:

- Datafil klargøring og vedligeholdelse.
- Indexfil klargøring.
- Indexfil vedligeholdelse.
- Indexfil søgning.

Rutinerne i hver af disse kategorier er opsummeret i de følgende afsnit. Beskrivelserne angiver tillige hvilket PolyFile modul, der skal inkluderes i programmet, for at gøre en given funktion tilgængelig.

1.2.1 Datafil klargøring og vedligeholdelse

makefile	Opretter en ny datafil (DATMAN).
openfile	Åbner en eksisterende datafil (DATMAN).
closefile	Lukker en datafil (DATMAN).
addrec	Opretter en ny datapost. addrec genbruger automatisk eventuelle slettede dataposter, før datafilen udvides (DATMAN).
deleterec	Sletter en datapost. Den slettede datapost bliver indsat i kæden af slettede poster, således at den kan genbruges af addrec, før datafilen udvides (DATMAN).
getrec	Læser en datapost (DATMAN).
putrec	Skriver en datapost (DATMAN).
filelen	Returnerer det totale antal poster i en datafil, inklusive brugte, slettede og reserverede poster (DATMAN).
usedrecs	Returnerer antallet af brugte poster i en datafil (DATMAN).

1.2.2 Indexfil klargøring

initindex	Initialiserer PolyFile indexsystemet (DATMAN).
makeindex	Opretter en ny indexfil (DATMAN).
openindex	Åbner en eksisterende indexfil (DATMAN).
closeindex	Lukker en indexfil (DATMAN).

1.2.3 Indexfil vedligeholdelse

addkey	Opretter en ny nøgle med et tilhørende datapost nummer i en indexfil (ADDKEY).
deletekey	Sletter en nøgle og dens tilhørende datapost nummer fra en indexfil (DELKEY).

1.2.4 Index fil søgning

findkey — Finder en given nøgle i en indexfil (NPFKEY).

searchkey — Finder den første nøgle i en indexfil der er større end eller lig med en given søgenøgle (NPFKEY).

nextkey — Returnerer den næste nøgle i en indexfil (NPFKEY).

prevkey — Returnerer den forrige nøgle i en indexfil (NPFKEY).

clearkey — Anbringer indexpilen ved starten eller slutningen af en indexfil, således at denne kan gennemløbes sekventielt (DATMAN).

1.3 PolyFile's typer og variable

Udover de grundlæggende data/index fil rutiner, definerer DATMAN modulet en række globale typer og variable. Disse er:

datafile — Denne typeidentifier bruges til at erklære datafil-variable. Alle PolyFile datafiler erklæres ved hjælp af denne identifier, selvom deres dataposter ikke er af same type og størrelse.

indexfile — Denne typeidentifier bruges til at erklære indexfil-variable.

ok — ok er en logisk (boolean) variabel, der bruges til at returnere status fra visse PolyFile rutiner.

1.4 Generelt om PolyFile

PolyFile's datafiler kan indeholde op til 65535 poster og poststørrelsen kan (i teorien) være op til 64K bytes. PolyFile datafilvariable erklæres altid ved hjælp af typen datafile, i modsætning til almindelige PolyPascal datafiler. Den aktuelle poststørrelse oplyses først under kørsel af programmet. Datafilerne har altid fast længde poster, og den mindste poststørrelse er 8 bytes. Den første post i en datafil (post nummer 0) er forbeholdt intern brug, og reserveres automatisk af addrec rutinen.

PolyFile ordner indexfiler efter en højdebalanceret flervejs træstruktur, også kaldet et B–træ. Denne organisering sikrer et minimum af

disklæsninger til gennemsøgning af et index, og medfører desuden, at indexfiler aldrig skal re–organiseres. PolyFile justerer automatisk indexfilen, når en nøgle oprettes eller slettes.

Indexfiler indeholder nøgleværdier (strenge) og datapostnumre, der angiver positionen af den pågældende datapost i datafilen. Ved hjælp af en nøgleværdi (eller blot en del af en sådan) er det således muligt at finde den tilhørende datapost.

PolyFile antager intet om sammenhængen mellem indexfiler og datafiler. Dermed menes, at det er op til programmet at trække nøgleinformationen ud af dataposten og aflevere den til indexopbygningsrutinen sammen med datapostnummeret. Af denne grund kan flere indexfiler eventuelt referere til samme datafil.

Det anbefales at anbringe nøgleinformationerne i både datafilen og indexfilerne, da det således er muligt at rekonstruere en ødelagt indexfil. Desuden kan programmet da anvende "beregnede" nøgler, dvs. nøgler, der er en funktion af et eller flere felter i dataposterne. Et eksempel: Antag at man ønsker at opbygge og vedligeholde et kundekartotek, der indexeres på kundernes efternavn. Nøglerne i indexfilen vil da være efternavnet fra hver post konverteret til store bogstaver. Hvis man ved en søgning også konverterer søgenøglen til store bogstaver, bliver man i stand til at finde den ønskede post, selvom man taster store bogstaver i stedet for små (eller omvendt), men når man læser dataposten, fremstår navnet præcis som det blev indtastet. Almindeligvis bør nøgleinformationer kun udelades fra dataposterne hvis der er begrænset plads på disken.

1.5 Brug af PolyFile

Som tidligere nævnt gøres PolyFile rutinerne tilgængelige ved at inkludere et eller flere af de fire PolyFile moduler. DATMAN modulet skal altid inkluderes før de andre PolyFile moduler, da det indeholder de grundlæggende rutiner. Efter DATMAN kan andre moduler (NPFKEY, ADDKEY og DELKEY) inkluderes efter behov.

Før DATMAN inkluderes, skal visse heltalskonstanter erklæres. Disse er:

maxrsize — Den maksimale poststørrelse. maxrsize skal sættes til størrelsen (i bytes) af den største datapost, programmet skal kunne behandle. Hvis programmmet for eksempel

skal behandle to datafiler med poster på 72 og 140 bytes, skal maxrsize sættes til 140.

maxksize — Den maksimale nøglelængde (et heltal mellem 1 og 255). maxksize skal sættes til længden af den største nøglestreng i de indexfiler, programmet skal behandle. Hvis programmet for eksempel skal behandle tre indexfiler med nøglestrenge på op til 16, 10 og 25 tegn, skal maxksize sættes til 25.

nodesize — Knudestørrelse. Det største antal nøgler, der tillades i en enkelt post (knude) i en indexfil. nodesize skal være den samme for alle indexfiler, der behandles af programmet. nodesize skal desuden være et lige heltal mellem 4 og 254. Yderligere beskrivelse følger nedenfor.

nodehalf — Halvdelen af nodesize.

nbufsize — Knudebufferstørrelse. Denne konstant definerer antallet af indexfilposter (knuder), der kan holdes i lageret på en gang. Den mindst tilladelige værdi er 3. Når nbufsize gøres større, stiger hastigheden af søgninger i indexfiler, idet sansynligheden for, at en indexfilpost allerede er i lageret bliver større.

maxdepth — Den maksimale "dybde" af en indexfilstruktur. Værdien gælder for alle index filer, programmet skal behandle. Yderligere beskrivelse følger nedenfor.

Det største antal indexknuder, k, der skal gennemsøges for at finde en given nøgle i en indexfil med e nøgler er:

```
k = log(e) / log(nodesize * 0.5)
```

hvor nodesize er knudestørrelsen. Heraf ses, at jo større knuderne er, jo færre disklæsninger (og dermed mindre tid) kræves der for at gennemsøge et index (den tid, det tager at gennemsøge en knude, når den først er i lageret, er ubetydelig i forhold til den tid, det tager at læse knuden fra disken). maxdepth konstanten, der forventes erklæret før DATMAN inkluderes, svarer til heltalsdelen af k plus 1, så når knudestørrelsen og det maksimale antal dataposter er fastlagt, kan maxdepth beregnes af:

```
maxdepth = int(log(e) / log(nodesize * 0.5)) + 1
```

Bemærk, at større værdier af maxdepth ikke kræver nævneværdigt ekstra lager. Det anbefales derfor at runde maxdepth op med 2 eller 3 i stedet for 1 for at være på den sikre side.

Antallet af bytes, n, der optages af hver knude i en indexfil er:

$$n = (ksize + 5) * nodesize + 3$$

hvor ksize er den maksimale nøglelængde for den pågældende indexfil. Det største antal bytes, d, en indexfil kan optage er:

$$d = n * e / (nodesize * 0.5)$$

Dette er det værst tænkelige tilfælde. Almindeligvis er faktoren for knudestørrelsen (nodesize) snarere 0.75 end de opgivne 0.5.

Det antal bytes, m, der optages i lageret af PolyFile's knudebuffer er:

$$m = (((maxksize + 5) * nodesize) + 3) * nbufsize$$

hvor maxksize er den største nøglelængde, der anvendes af de indexfiler, der skal behandles, og nbufsize er det maksimale antal knuder der kan holdes i lageret på en gang (mindst 3).

Det er svært at anvise en generel metode til beregning af de optimale værdier for nodesize og nbufsize.

nodesize ligger i regelen mellem 16 og 32, afhængig af den maksimale nøglelængde og antallet af nøgler i indexfilen. Mindre værdier resulterer i unødig langsomme søgninger, medens større værdier kræver for megen plads til knudebufferen.

Den mindste fornuftige værdi nbufsize kan antage, er værdien af maxdepth. Hvis nbufsize er mindre en maxdepth, risikerer man, at samme knude skal læses flere gange for at gennemløbe indexfilen. Generelt bør nbufsize være så stor som muligt. Bemærk især, at når nbufsize er større end nodesize, kan PolyFile holde både roden og hele det første niveau af B–Træet i lageret, hvilket reducerer antallet af disklæsninger med mindst en for hver søgning.

Det generelle udseende af et program, der bruger PolyFile, er:

```
PROGRAM generelt;

CONST
  maxrsize = 132;          { Maksimal poststørrelse }
  maxksize = 25;           { Maksimal nøglelængde }
  nodesize = 24;           { Knudestørrelse }
  nodehalf = 12;           { Knudestørrelse/2 }
  nbufsize = 8;            { Knudebufferstørrelse }
  maxdepth = 5;            { Maksimal indexdybde }

{$I DATMAN} {              Skal komme før andre moduler }

{ Inkludering af andre PolyFile moduler }

{ Egne erklæringer }

BEGIN

  { Hovedprogram }

END.
```

1.6 Programstruktur med PolyFile

Et program, der anvender PolyFile, vil i regelen udføre en eller flere af følgende fire funktioner: Opret datapost, læs datapost, opdater datapost, og slet datapost. Desuden vil et program, før det behandler data, åbne de nødvendige index- og datafiler, og før det slutter, lukke dem igen.

Filklargøringen består af kald til makefile eller openfile for hver datafil, der skal behandles, og kald til makeindex eller openindex for hver indexfil, der skal behandles. Desuden skal der udføres et kald til initindex for at klargøre indexfil rutinerne (dette kald udføres typisk i begyndelsen af programmet, og det er kun nødvendigt hvis programmet anvender indexfiler).

For at oprette en ny datapost skal programmet først kalde addrec, hvorved dataposten indsættes i datafilen. Dernæst beregnes en nøgle-

værdi (ofte blot et enkelt felt fra dataposten), og denne indsættes i indexfilen ved et kald til addkey. Ved kaldet til addkey skal programmet overføre det postnummer, addrec returnerede, således at det sammen med nøglen kan indgå i indexfilen. Hvis der er mere end en indexfil, skal der udføres et kald til addkey for hver indexfil, med det samme postnummer.

En given datapost fremfindes ved først at udregne dens nøgleværdi, og derefter at gennemsøge indexfilen (eller en af indexfilerne) ved hjælp af findkey, searchkey, nextkey, prevkey, og clearkey rutinerne. Når den ønskede nøgle er fundet, bruges getrec til at læse den tilhørende datapost fra datafilen.

Når et datapost skal slettes, må man først bestemme dens nøgleværdi, for eksempel ved kald til findkey, searchkey, nextkey, prevkey, og clearkey. Derefter bruges deletekey til at slette nøglen. deletekey returnerer det datapostnummer, der hører sammen med den slettede nøgle. Hvis programmet behandler flere indexfiler, kan dette nummer bruges til at læse dataposten (getrec), hvorefter nøglerne i de andre indexfiler kan beregnes og slettes. Til sidst kaldes deleterec for at slette dataposten.

Opdatering af en datapost kan til tider medføre, at dens nøgleværdi (eller nøgleværdier) ændrer sig. I så tilfælde bruges deletekey til at slette den gamle nøgle, og addkey til at oprette den nye. Hvis der er flere indexfiler, gentages dette for alle de filer, der berøres af ændringen. Til sidst bruges putrec til at udskrive den ændrede datapost.

Når et program slutter, skal det kalde closefile for hver datafil, der har været i brug, og closeindex for hver indexfil.

2. PolyFile rutiner

Dette kapitel beskriver de rutiner, PolyFile gør tilgængelige for PolyPascal programmøren.

Størstedelen af PolyFile rutinerne returnerer deres status i en logisk variabel ved navn ok. Denne variabel erklæres automatisk af PolyFile. For eksempel sætter openfile rutinen ok til true (sand) hvis filen findes og false (falsk) hvis ikke. Hvis der opstår alvorlige og/eller uoprettelige fejl, kaldes proceduren iocheck i DATMAN modulet. Normalt udskriver iocheck navnet på filen, postnummeret, og fejlkoden, hvorefter programmet stoppes. Man kan imidlertid skrive sin egen iocheck procedure, hvilket omtales i kapitel 4.

2.1 Datafil vedligeholdelsesrutiner

PolyFile's datafil vedligeholdelsesrutiner omfatter:

- Oprettelse, åbning, og lukning af datafiler.
- Oprettelse og sletning af dataposter i datafiler.
- Læsning og skrivning af dataposter fra/til datafiler.
- Beregning af størrelsen af datafiler.

Alle rutiner, der beskrives i afsnit 2.1, er indeholdt i DATMAN modulet.

PolyFile datafilvariable erklæres altid ved hjælp af typen datafile, i modsætning til almindelige PolyPascal datafiler. Den aktuelle poststørrelse oplyses først under kørsel af programmet. Datafilerne har altid fast længde poster. Den mindst tilladelige poststørrelse er 8 bytes, men der er ingen øvre grænse. Den første post i en datafil (post nummer 0) er forbeholdt intern intern brug, og reserveres automatisk af addrec rutinen.

Som tidligere nævnt, genbruger PolyFile automatisk slettede dataposter, før datafilen udvides, når der oprettes nye poster. Dette opnås ved at opbygge en kædet liste af frie poster: Når en post er slettet, udgør de to første bytes i posten en pointer til den næste slettede post (–1 angiver, at posten er den sidste i listen). Da der aldrig forekommer et nul i pointeren, kan (og bør) man reservere de to første bytes i hver

post, og sætte dem til nul når en post oprettes. Man kan da se forskel på brugte og slettede poster uden at anvende en indexfil (dette er en nødvendighed, hvis man skal rekonstruere en ødelagt indexfil).

2.1.1 makefile rutinen

```
PROCEDURE makefile(VAR datf: datafile; fnam: fnstr;
reclen: integer);
```

makefile opretter en ny datafil og klargør den til behandling af PolyFile rutinerne. Parametrene til makefile er:

datf	Datafilvariablen, der skal klargøres til brug. datf skal være af typen datafile.
fnam	Et strengudtryk, der angiver navnet på den nye datafil.
reclen	Poststørrelsen i bytes (mindst 8).

makefile sætter ok til sand hvis filen kunne oprettes og klargøres. ok er falsk hvis der ikke er plads til at oprette en ny fil på disken.

2.1.2 openfile rutinen

```
PROCEDURE openfile(VAR datf: datafile; fnam: fnstr;
reclen: integer);
```

openfile åbner en eksisterende datafil og klargør den til behandling af PolyFile rutinerne. Parametrene til openfile er:

datf	Datafilvariablen, der skal klargøres til brug. datf skal være af typen datafile.
fnam	Et strengudtryk, der angiver navnet på en eksisterende datafil.
reclen	Poststørrelsen i bytes. Poststørrelsen skal være den samme som den, der blev anvendt da filen blev oprettet.

openfile sætter ok til sand hvis filen kunne åbnes. Ellers sættes ok til falsk.

2.1.3 closefile rutinen

PROCEDURE closefile(VAR datf: datafile);

closefile lukker datafilen givet ved datf. closefile skal altid kaldes, hvis der er foretaget ændringer i en datafil. Ellers risikerer man at der går data tabt, og at pointerkæden for slettede poster ødelægges.

2.1.4 addrec rutinen

PROCEDURE addrec(VAR datf: datafile; VAR drn: integer; VAR buffer);

addrec — opretter en ny datapost i en PolyFile datafil. Hvis der er tidligere brugte poster til rådighed i filen, anvendes disse før filen udvides. Parametrene til addrec er:

datf — Datafilen hvori posten skal oprettes.

drn — Datapostnummeret. addrec returnerer datapostnummeret i denne var–parameter. Datapostnummeret skal overføres til addkey, når man opretter en nøgle for posten.

buffer — En variabel, der indeholder den datapost, der skal oprettes. Da buffer er en typeløs parameter, accepterer addrec enhver variabel i dens sted. Det er op til programmøren at sikre, at den variabel, der overføres, er af den rigtige type.

addrec returnerer ikke status i ok variablen. Proceduren returnerer kun hvis dataposten blev oprettet uden fejl. I tilfælde af fejl kaldes iocheck proceduren, og denne stopper programmet. Det er en god ide at kalde usedrecs før addrec, for at sikre, at der er plads til yderligere poster.

2.1.5 deleterec rutinen

PROCEDURE deleterec(VAR datf: datafile; drn: integer);

deleterec sletter en datapost i en PolyFile datafil. Den slettede post indsættes i listen af slettede poster, således at den senere kan genbruges af addrec. Parametrene til deleterec er:

datf — Datafilen hvori posten skal slettes.

drn — Datapostnummeret. Nummeret på den post, der skal slettes. Datapostnummeret skal også angives når deletekey kaldes for at slette datapostens nøgle.

2.1.6 getrec rutinen

PROCEDURE getrec(VAR datf: datafile; drn: integer;
VAR buffer);

getrec læser en datapost fra en given position i en datafil. Parametrene til getrec er:

datf — Datafilen hvorfra posten skal læses.

drn — Datapostnummeret.

buffer — Den variabel posten skal læses ind i. Da buffer er en typeløs parameter, accepterer addrec enhver variabel i dens sted. Det er op til programmøren at sikre, at den variabel, der overføres, er af den rigtige type.

2.1.7 putrec rutinen

PROCEDURE putrec(VAR datf: datafile; drn: integer;
VAR buffer);

putrec skriver en datapost til en given position i en datafil. Parametrene til putrec er:

datf — Datafilen hvori posten skal skrives.

drn — Datapostnummeret.

buffer — En variabel der indeholder dataposten. Da buffer er en

typeløs parameter, accepterer addrec enhver variabel i dens sted. Det er op til programmøren at sikre, at den variabel, der overføres, er af den rigtige type.

2.1.8 filelen functionen

FUNCTION filelen(VAR datf: datafile): integer;

filelen returnerer det total antal dataposter i datafilen datf. Denne værdi omfatter både slettede poster og den reserverede post (post nummer 0).

2.1.9 usedrecs functionen

FUNCTION usedrecs(VAR datf: datafile): integer;

usedrecs returnerer antallet af brugte poster i datafilen datf. I modsætning til filelen omfatter denne værdi ikke slettede og reserverede poster.

2.2 Indexfil vedligeholdelsesrutiner

PolyFile's indexfil vedligeholdelsesrutiner omfatter:

- Oprettelse, åbning, og lukning af indexfiler.
- Oprettelse og sletning af nøgler i indexfiler.
- Søgning i indexfiler.
- Forlæns og baglæns gennemløb af indexfiler.

Indexfiler erklæres ved hjælp af typen indexfile. Elementerne i en indexfil består af en nøglestreng og et datapostnummer. Nøglestrenge kan antage værdier af typen string. Den maksimale længde af nøglerne angives når indexfilen åbnes. Hvis en nøglestreng er for lang, bliver den afkortet såsnart den overgives til en indexfil rutine.

Dublerede nøgler

I mange tilfælde er det en fordel at kunne arbejde med nøgler, der ikke er entydige. Et typisk eksempel er et index, der er baseret på navne. PolyFile tillader kun dublerede nøgler hvis status parametren i kaldet til makeindex eller openindex er 1.

Når PolyFile opretter en ny nøgle i et index, der tillader dublerede

nøgler, bliver sammenfaldende nøgler ordnet efter deres tilhørende datapostnumre, således at de nøgler, der har de laveste datapostnumre, kommer først. Normalt svarer dette til den rækkefølge hvori nøglerne blev oprettet, da dataposter oftest oprettes i enden af datafilerne.

Søgerutinerne (findkey og searchkey) finder altid den første nøgle, dvs. den nøgle, der har det laveste datapostnummer.

Når en nøgle skal slettes fra en indexfil med dublerede nøgler, skal både nøglen og datapostnummeret opgives, da nøglen alene kan udpege flere indgange. deletekey rutinen sletter kun indgangen hvis både nøglen og datapostnummeret svarer til de opgivne værdier.

Numeriske nøgler

Hvis et program skal indexere på numeriske værdier, skal værdierne konverteres til strenge, før de overgives til PolyFile. Dette kan gøre på to måder.

Nemmest er at anvende PolyPascal's standardprocedure str til at konvertere tallet til en streng. Det er vigtigt at talstrengen er højrejusteret (dette opnås ved at angive en feltbredde ved konverteringen). Ulempen ved denne metode er, at nøglelængden skal sættes til største antal cifre, der kan forekomme i talstrengen, hvilket ofte fylder mere end tallets binære format.

Den alternative metode (der kun kan anvendes på heltal) udnytter den kompakte binære repræsentation. De nedenfor viste rutiner kan anvendes til at "pakke" og "udpakke" heltal til/fra strenge. itos konverterer et heltal til en streng, og stoi konverterer den anden vej. Strengene er altid to tegn lange.

```
FUNCTION itos(n: integer): str2;
BEGIN
  n:=n+$8000; itos:=chr(hi(n))+chr(lo(n));
END;

FUNCTION stoi(s: str2): integer;
BEGIN
  n:=swap(ord(s[1]))+ord(s[2])+$8000;
END;
```

Rutinerne arbejder på heltal med fortegn (–32768..32767). Hvis heltallene skal opfattes som fortegnsløse værdier, skal additionen af $8000 fjernes.

Opdeling af datafiler
En PolyFile indexfil skal være indeholdt i en enkelt diskfil. Derimod kan et datasæt godt deles over flere datafiler, dog med den begrænsning, at der ikke kan være mere en 65535 poster ialt. Opdelingen af datafilerne er ganske simpel at opnå og kan bedst illustreres ved et eksempel: Antag at hver datafil kan indeholde 10000 poster og at man ønsker at lagre op til 30000 poster. Der skal da anvendes tre datafiler. Når man opretter en post i den første datafil, bliver datapostnummeret indsat i indexfilen som det er. Når en post oprettes i den anden eller tredje datafil, adderes 10000 eller 20000 til datapostnummeret, før det indsættes i indexfilen. Når man senere læser indexfilen, vil en division med 10000 afgøre hvilken datafil, man kan finde dataposten i, medens resten fra divisionen angiver datapostnummeret. Bemærk at det første brugbare datapostnummer i en datafil er 1. Man må derfor trække 1 fra før divisionen, og lægge 1 til resten efter divisionen for at få korrekte resultater.

2.2.1 initindex rutinen

```
PROCEDURE initindex;
```

initindex klargør de interne tabeller, der anvendes af indexfil rutinerne. initindex skal kaldes før man begynder at anvende indexfil rutinerne. Det er nok med et enkelt kald, og dette udføres normalt i begyndelsen af programmet, der bruger PolyFile.

2.2.2 makeindex rutinen

```
PROCEDURE makeindex(VAR idxf: indexfile; fnam: fnstr;
keylen,status: integer);
```

makeindex opretter en ny indexfil og klargør den til behandling af PolyFile rutinerne. Parametrene til makeindex er:

idxf	Indexfilvariablen, der skal klargøres til brug. idxf skal være af typen indexfile.
fnam	Et strengudtryk, der angiver navnet på den nye datafil.

keylen — Den største længde nøglestrengene i indexfilen kan antage.

status — Filstatus. 0 angiver, at dublerede nøgler ikke tillades. 1 angiver, at dublerede nøgler må forekomme.

makeindex sætter ok til sand hvis filen kunne oprettes og klargøres. ok er falsk hvis der ikke er plads til at oprette en ny fil på disken.

2.2.3 openindex rutinen

```
PROCEDURE openindex(VAR idxf: indexfile; fnam: fnstr;
keylen,status: integer);
```

openindex åbner en eksisterende index fil og klargør den til behandling af PolyFile rutinerne. Parametrene til openindex er:

idxf — Indexfilvariablen, der skal klargøres til brug. idxf skal være af typen indexfile.

fnam — Et strengudtryk, der angiver navnet på en eksisterende datafil.

keylen — Den største længde nøglestrengene i indexfilen kan antage. keylen's værdi skal være den samme som den, der blev angivet ved makeindex.

status — Filstatus. 0 angiver, at dublerede nøgler ikke tillades. 1 angiver, at dublerede nøgler må forekomme.

openfile sætter ok til sand hvis filen kunne åbnes. Ellers sættes ok til falsk.

2.2.4 closeindex rutinen

```
PROCEDURE closeindex(VAR idxf: indexfile);
```

closeindex lukker indexfilen givet ved idxf. closeindex skal altid kaldes, hvis der er foretaget ændringer i en indexfil. Ellers risikerer man at der går data tabt, og at indexfilens struktur ødelægges.

2.2.5 addkey rutinen

```
PROCEDURE addkey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

addkey opretter en ny nøgle i en indexfil. ADDKEY modulet skal inkluderes for at addkey rutinen kan anvendes. Parametrene til addkey er:

idxf — Indexfilen hvori nøglen skal oprettes.

drn — Datapostnummeret, der skal høre til nøglen. drn er sædvanligvis et datapostnummer, der er blevet returneret af addrec.

key — Nøglen, der skal oprettes. Da key er en typeløs parameter, accepterer addkey enhver variabel i dens sted. Det er op til programmøren at sikre, at den variabel, der overføres, er en streng af den rigtige type. Hvis nøglestrengen er længere end den maksimale nøglelængde for indexfilen idxf, bliver den afkortet til maksimallængden.

addkey sætter ok til sand hvis nøglen blev oprettet uden fejl. ok sættes til falsk hvis man prøver at oprette en dubleret nøgle når en sådan ikke tillades (dvs. når status var 0 i kaldet til makeindex eller openindex).

Et kald til addkey medfører at indexpilen nulstilles, svarende til et kald til clearkey. Det er derfor ikke muligt at oprette nye nøgler samtidig med at man gennemløber et index ved kald til nextkey eller prevkey rutinerne.

2.2.6 deletekey rutinen

```
PROCEDURE deletekey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

deletekey sletter en nøgle i en indexfil. DELKEY modulet skal inkluderes for at deletekey rutinen kan anvendes. Parametrene til deletekey er:

idxf — Indexfilen hvorfra nøglen skal fjernes.

drn — Datapostnummeret, der hører til den nøgle, der skal

slettes. drn's værdi anvendes ikke, hvis indexfilen ikke tillader dublerede nøgler. Hvis dublerede nøgler tillades må deletekey nødvendigvis kende datapostnummeret, da det ellers ikke er muligt at skelne mellem ens nøgler. Datapostnummeret kan eventuelt bestemmes ved hjælp af searchkey, nextkey, og prevkey. drn returnerer altid datapostnummeret fra den nøgle, der blev slettet.

key — Nøglen, der skal slettes. Hvis nøglestrengen er længere end den maksimale nøglelængde for indexfilen idxf, bliver den afkortet til maksimallængden.

deletekey sætter ok til sand hvis nøglen blev fundet og fjernet uden fejl. Ellers sættes ok til falsk. Hvis dublerede nøgler tillades, returnerer ok falsk hvis datapostnummeret ikke passede, selvom nøglen fandtes.

Et kald til deletekey medfører at indexpilen nulstilles, svarende til et kald til clearkey. Det er derfor ikke muligt at slette nøgler samtidig med at man gennemløber et index ved kald til nextkey eller prevkey rutinerne.

2.2.7 findkey rutinen

```
PROCEDURE findkey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

findkey returnerer det datapostnummer, der hører til en given nøgle i en indexfil. NPFKEY modulet skal inkluderes for at nextkey rutinen kan anvendes. findkey returnerer den nøgle i indexfilen, der nøjagtigt svarer til søgenøglen. Hvis indexfilen indeholder dublerede nøgler, returnerer findkey altid den første nøgle. Parametrene til findkey er:

idxf — Indexfilen, der skal søges i.

drn — Hvis nøglen findes, returneres dens tilhørende data postnummer i denne parameter.

key — Nøglen, der skal søges efter. Hvis nøglestrengen er længere end den maksimale nøglelængde for indexfilen idxf, bliver den afkortet til maksimallængden.

findkey sætter ok til sand hvis nøglen blev fundet. Ellers sættes ok til falsk.

2.2.8 searchkey rutinen

```
PROCEDURE searchkey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

searchkey returnerer det datapostnummer, der hører til den første nøgle i en indexfil, der er større end eller lig med søgenøglen. NPFKEY modulet skal inkluderes for at searchkey rutinen kan anvendes. searchkey kan for eksempel bruges til en søgning hvor man kun kender den første del af nøglen. Hvis indexfilen indeholder dublerede nøgler, returnerer searchkey altid den første nøgle. Parametrene til searchkey er:

idxf — Indexfilen, der skal søges i.

drn — Hvis en nøgle findes, returneres dens tilhørende datapostnummer i denne parameter.

key — Nøglen, der skal søges efter. Hvis nøglestrengen er længere end den maksimale nøglelængde for indexfilen idxf, bliver den afkortet til maksimallængden.

searchkey sætter altid ok til sand, medmindre der ikke er nogen nøgler, der er større end eller lig den angivne søgenøgle. I sidstnævnte tilfælde sættes ok til falsk.

2.2.9 nextkey rutinen

```
PROCEDURE nextkey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

nextkey returnerer nøglen og datapostnummeret for den næste indgang i indexfilen. NPFKEY modulet skal inkluderes for at nextkey rutinen kan anvendes. Parametrene til nextkey er:

idxf — En indexfil, der er forberedt for sekventiel behandling ved et kald til findkey, searchkey eller clearkey.

drn — Returnerer datapostnummeret fra den næste indgang i indexfilen.

key — Returnerer nøglestrengen fra den næste indgang i indexfilen.

nextkey sætter ok til sand, medmindre der ikke er en næste indgang i indexfilen. Når ok er falsk, dvs. når indexpilen er ved slutningen af indexfilen, vil det næste kald til nextkey returnere den første nøgle i indexfilen.

nextkey bruges normalt efter et kald til findkey, searchkey, eller clearkey. Bemærk, at kald til addkey og deletekey medfører, at indexpilen nulstilles, svarende til et kald til clearkey.

2.2.10 prevkey rutinen

```
PROCEDURE prevkey(VAR idxf: indexfile;
VAR drn: integer; VAR key);
```

prevkey returnerer nøglen og datapostnummeret for den forrige indgang i indexfilen. NPFKEY modulet skal inkluderes for at prevkey rutinen kan anvendes. Parametrene til prevkey er:

idxf	En indexfil, der er forberedt for sekventiel behandling ved et kald til findkey, searchkey eller clearkey.
drn	Returnerer datapostnummeret fra den forrige indgang i indexfilen.
key	Returnerer nøglestrengen fra den forrige indgang i indexfilen.

prevkey sætter ok til sand, medmindre der ikke er en forrige indgang i indexfilen. Når ok er falsk, dvs. når indexpilen er ved starten af indexfilen, vil det næste kald til prevkey returnere den sidste nøgle i indexfilen.

prevkey bruges normalt efter et kald til findkey, searchkey, eller clearkey. Bemærk, at kald til addkey og deletekey medfører, at indexpilen nulstilles, svarende til et kald til clearkey.

2.2.11 clearkey rutinen

```
PROCEDURE clearkey(VAR idxf: indexfile);
```

clearkey sætter indexpilen til at pege på starten eller slutningen af indexfilen. Efter et kald til clearkey, returnerer et kald til nextkey den første indgang i indexfilen, og et kald til prevkey den sidste indgang i indexfilen.

Når en PolyFile indexfil behandles sekventielt, kan den betragtes som en ring. Når indexpilen er ved slutningen af indexfilen, returnerer et kald til nextkey den første indgang filen, og når indexpilen er ved starten af indexfilen, returnerer et kald til prevkey den sidste indgang filen. Starten og slutningen af filen er faktisk på samme sted.

3. Programeksempler

Eksemplerne i dette kapitel viser nogle typiske operationer på en PolyFile database. Databasen, der her anvendes, er ganske simpel. Dataposterne har kun to felter, og der er kun en indexfil. Principperne er imidlertid de samme for databaser med større poster og flere indexfiler.

Datafilens poster er af følgende type:

```
TYPE
  phonerec = RECORD
               status: integer;
               phone: STRING[15];
               name: STRING[30];
             END;
```

phone feltet bruges til at indexere datafilen. Bemærk status feltet i starten af posten. Hvis dette felt sættes til nul når en ny post oprettes, kan det bruges til at bestemme om en post er slettet eller ej, når datafilen gennemlæses uden reference til indexfilen (for eksempel ved genopbygning af en ødelagt indexfil).

PolyFile konstanterne defineres som:

```
CONST
  maxrsize = 49;          { Maksimal poststørrelse }
  maxksize = 15;          { Maksimal nøglelængde }
  nodesize = 16;          { Knudestørrelse }
  nodehalf = 8;           { Knudestørrelse/2 }
  nbufsize = 5;           { Knudebufferstørrelse }
  maxdepth = 5;           { Maksimal indexdybde }
```

maxrsize er størrelsen af phonerec typen, og maxksize er maksimallængden af phone feltet.

De variable, der bruges i eksemplerne, er defineret som:

```
VAR
  datf: datafile;
  idxf: indexfile;
  prec: phonerec;
  pn: STRING[15];
  ch: char;
  d: integer;
```

Eksemplet herunder opretter en datafil kaldet PHONE.DAT og en indexfil kaldet PHONE.IDX. Bemærk brugen af size funktionen til at bestemme poststørrelsen.

```
makefile(datf,'PHONE.DAT',size(prec));
IF ok THEN makeindex(idxf,'PHONE.IDX',15,0);
IF ok THEN
BEGIN
  closefile(datf); closeindex(idxf);
END ELSE
writeln('Cannot create data files');
```

Det næste eksempel viser hvorledes man opretter nye poster i databasen. Da dublerede nøgler ikke tillades, sættes status parametren i kaldet af openindex til 0. En ny post kan kun oprettes hvis et kald til findkey returnerer falsk i ok (hvilket indikerer, at den pågældende nøgle ikke allerede findes). Hvis programmet straks kaldte addrec, måtte det derefter kalde deleterec, hvis det viste sig, at nøglen allerede fandtes (ellers ville databasen indeholde en post, der ikke var i indexet). Bemærk, at status feltet nulstilles for at markere, at posten er i brug.

```
initindex;
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN openindex(idxf,'PHONE.IDX',15,0);
IF ok THEN
BEGIN
  REPEAT
    write('Phone number....: '); readln(pn);
    prec.phone:=pn; findkey(idxf,d,pn);
    IF ok THEN
    BEGIN
      writeln; writeln('Already assigned');
```

```
      END ELSE
      BEGIN
        write('Name.....: '); readln(prec.name);
        prec.status:=0;
        addrec(datf,d,prec); addkey(idxf,d,prec.phone);
      END;
      writeln;
      write('Add another (Y/N)? '); readln(ch);
      writeln;
    UNTIL NOT(ch IN ['Y','y']);
    closefile(datf); closeindex(idxf);
  END ELSE
  writeln('Cannot open data files');
```

Det følgende eksempel viser hvordan man finder poster ved hjælp af indexfilen. findkey bruges til at søge i indexfilen, og hvis den givne nøgle findes, returneres datapostnummeret. Dette nummer bruges derefter i et kald til getrec, der læser dataposten.

```
initindex;
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN openindex(idxf,'PHONE.IDX',15,0);
IF ok THEN
BEGIN
  REPEAT
    write('Phone number...: '); readln(pn);
    findkey(idxf,d,pn);
    IF ok THEN
    BEGIN
      getrec(datf,d,prec);
      writeln('Name.....: ',prec.name);
    END ELSE
    BEGIN
      writeln; writeln('Key not found');
    END;
    writeln;
    write('Scan again (Y/N)? '); readln(ch);
    writeln;
  UNTIL NOT(ch IN ['Y','y']);
  closefile(datf); closeindex(idxf);
END ELSE
writeln('Cannot open data files');
```

Det næste eksempel viser hvordan man sletter poster. deletekey bruges til at slette nøglen, og hvis dette går godt, dvs. hvis nøglen eksisterede, bruges det datapostnummer, der returneres, i et kald til deleterec, hvilket sletter selve dataposten.

```
initindex;
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN openindex(idxf,'PHONE.IDX',15,0);
IF ok THEN
BEGIN
  REPEAT
    write('Phone number....: '); buflen:=15; readln(pn);
    deletekey(idxf,d,pn);
    IF ok THEN deleterec(datf,d) ELSE
    BEGIN
      writeln; writeln('Key not found');
    END;
    writeln;
    write('Scan again (Y/N)? '); readln(ch);
    writeln;
  UNTIL NOT(ch IN ['Y','y']);
  closefile(datf); closeindex(idxf);
END ELSE
writeln('Cannot open data files');
```

clearkey og nextkey rutinerne anvendes i det næste eksempel til at gennemlæse databasen sekventielt. Bemærk at nextkey returnerer falsk i ok når alle nøgler i indexfilen er behandlet.

```
initindex;
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN openindex(idxf,'PHONE.IDX',15,0);
IF ok THEN
BEGIN
  clearkey;
  REPEAT
    nextkey(idxf,d,pn);
    IF ok THEN
    BEGIN
      getrec(datf,d,prec);
      writeln(prec.phone,':16-len(prec.phone),prec.name);
    END;
  UNTIL NOT ok;
```

```
  closefile(datf); closeindex(idxf);
END ELSE
writeln('Cannot close data files');
```

En datafil kan også behandles uden reference til en indexfil. Dette kræver dog, at det første felt i hver post er et statusfelt (en integer). Når programmet opretter en ny post (ved hjælp af addrec rutinen), skal statusfeltet manuelt sættes til nul. Ved det sekventielle gennemløb af datafilen, der udføres af eksemplet nedenfor, vil poster, der er i brug, have værdien nul i deres statusfelt, medens slettede poster har en værdi forskellig fra nul. Bemærk, at læsningen starter ved post nummer 1, da post nummer 0 er reserveret af PolyFile til internt brug.

```
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN
BEGIN
  d:=1;
  WHILE d<filelen(datf) DO
  BEGIN
    getrec(datf,d,prec); d:=d+1;
    IF prec.status=0 THEN
    writeln(prec.phone,':16-len(prec.phone),prec.name);
  END;
  closefile(datf);
END ELSE
writeln('Cannot open data file');
```

Det sidste programeksempel viser hvordan man genopbygger en ødelagt indexfil. Programmet gennemlæser datafilen og opretter en nøgle for alle aktive poster i en ny indexfil. Der udskrives en fejlmeddelelse hvis to aktive poster har samme nøgle.

```
initindex;
openfile(datf,'PHONE.DAT',size(prec));
IF ok THEN
BEGIN
  makefile(idxf,'PHONE.IDX',15,0);
  IF ok THEN
  BEGIN
    d:=1;
    WHILE d<filelen(datf) DO
    BEGIN
      getrec(datf,d,prec);
      IF prec.status=0 THEN
      BEGIN
        addkey(idxf,d,prec.phone);
        IF NOT ok THEN
        BEGIN
          writeln('Duplicate key. Record ',d);
          deleterec(datf,d);
        END;
      END;
      d:=d+1;
    END;
  END ELSE
  writeln('Cannot create index file');
END ELSE
writeln('Cannot open data file');
```

4. PolyFile fejlhåndtering

PolyFile rutinerne definerer to typer fejl: Ikke–fatale og fatale. Fatale fejl medfører, at programmet stoppes, medens ikke–fatale fejl rapporteres til det kaldende program.

Ikke–fatale fejlsituationer rapporteres gennem den logiske variabel ok, der erklæres af DATMAN modulet. For eksempel returnerer openfile falsk i ok hvis den angivne fil ikke findes.

Hvis der opstår en fatal fejl, kaldes en rutine ved navn iocheck i DATMAN modulet. iocheck udskriver en fejlkode, et filnavn, og et datapostnummer, hvorefter programmet stoppes. Et eksempel:

```
PolyFile I/O Error 10
File A:DATABASE.DAT Record 103
Program terminated
```

En fatal fejl i PolyFile er i realiteten en PolyPascal I/O fejl. De mulige fejlkoder er derfor de samme som dem, der er vist i Appendix F i PolyPascal Programmeringsmanualen. Bemærk dog, at PolyFile udskriver fejlkoderne i decimal notation, medens PolyPascal udskriver dem i hexnotation.

Fatale fejl skyldes sædvanligvis ødelagte datafiler og/eller indexfiler. Der opstår imidlertid også en fatal fejl, hvis man prøver at udvide en datafil eller en indexfil, når der ikke er mere plads på disken.

Hvis man ønsker en mere kontrolleret programafbrydelse i tilfælde af en fatal fejl, kan man selv skrive en ny iocheck rutine. Dette gøres ved at ændre iocheck rutinen i DATMAN modulet til et FORWARD specificeret procedure, hvorefter selve rutinen kan følge i brugerprogrammet. Bemærk, at iocheck ikke må returnere til det kaldende program – rutinens eneste formål er at udskrive en fejlmeddelelse og stoppe programmet. PolyFile's tilstand er udefineret hvis iocheck returnerer, hvilket eventuelt kan gå ud over programmets filer.

En brugerdefineret iocheck rutine kan eventuelt lukke filer ved kald til closefile og closeindex før den afbryder programmet. Ved sådanne kald kan imidlertid opstå nye fejl, hvilket resulterer i yderligere kald

af iocheck, og i så tilfælde skal iocheck returnere umiddelbart for at undgå uendelige løkker. Det følgende eksempel viser hvordan dette kan opnås ved hjælp af en logisk variabel.

iocheck proceduren i DATMAN modulet skal ændres til en FORWARD specifikation for at implementere en brugerdefineret rutine:

```
PROCEDURE iocheck(VAR datf: datafile; r: integer);
FORWARD;
```

I brugerprogrammet anbringes de følgende erklæringer lige efter de compilerdirektiver, der inkluderer PolyFile modulerne:

```
VAR
  error: boolean;

PROCEDURE iocheck;
VAR
  p: integer;
BEGIN
  IF (iosts<>0) AND NOT error THEN
  BEGIN
    writeln;
    writeln('PolyFile I/O Error ',iosts);
    write('File '); p:=0;
    WHILE datf.fnam[p]<>@0 DO
    BEGIN
      write(datf.fnam[p]); p:=succ(p);
    END;
    writeln(' Record ',r);
    error:=true;
    IF datopen THEN closefile(datf);
    IF idxopen THEN closeindex(idxf);
    writeln('Program terminated');
    halt;
  END;
END;
```

Ovenstående eksempel er beregnet til MS–DOS versionen af PolyFile. I CP/M versionenerne skal erklæringen af p udelades og WHILE løkken erstattes med:

```
write(chr(datf.cdrv+64),':',datf.cnam,'.',datf.ctyp);
```

error variablen skal sættes til falsk (false) i begyndelsen af brugerprogrammet. Hvis iocheck kaldes på grund af en fejl, sættes error til sand (true), og yderligere fejl (på grund af kaldene til closefile og closeindex) ignoreres.

datf og idxf er i dette eksempel de filer, der bearbejdes af programmet, og det antages, at de er erklæret andetsteds. datopen og idxopen er logiske variable, der afspejler status af de respektive filer. Når en variabel er sand, antages det, at den tilsvarende fil er åben, og derfor skal lukkes hvis iocheck kaldes.

Index

A

addkey, 23
ADDKEY modulet, 7
addrec, 17

C

clearkey, 27
closefile, 17
closeindex, 22

D

Datafil rutiner, 15
datafile typen, 9
Datafiler, 9, 15
  erklæring, 9
DATMAN modulet, 7
deletekey, 23
deleterec, 18
DELKEY modulet, 7
Dublerede nøgler, 19

F

filelen, 19
findkey, 24
Funktionsoversigt, 7

G

getrec, 18

I

Indexfil rutiner, 19
indexfile typen, 9
Indexfiler, 9, 19
  erklæring, 9
initindex, 21

K

Konstanter, 10
  beregning af, 11

M

makefile, 16
makeindex, 21
maxdepth konstant, 11
maxksize konstant, 11
maxrsize konstant, 10

N

nbufsize konstant, 11
nextkey, 25
nodehalf konstant, 11
nodesize konstant, 11
NPFKEY modulet, 7
Numeriske nøgler, 20
Nøgler, 19
  dublerede, 19
  numeriske, 20

O

ok variablen, 9
Opdeling af datafiler, 21
openfile, 16
openindex, 22
Oversigt over funktioner, 7

P

prevkey, 26
Programstruktur, 13
putrec, 18

S

searchkey, 25

U

usedrecs, 19