INNEHÅLLSFÖRTECKNING

KAPITEL 0: INTRODUKTION

0.0 Att komma igång

Hisoft Pascal 4T (HP4T) är en snabb, användarvänlig och kraftfull version av Pascal, så som den är definierad i 'Pascal User Manual and Report' (Jensen/Wirth, andra upplagan). Avvikelserna från denna specifikation är följande:

FILE:er implementeras inte. Variabler kan dock lagras på kassettband. En RECORD får inte innehålla delar bestående av VARIANT. PROCEDURE och FUNCTION är inte giltiga parametrar.

Många extra funktioner och procedurer finns tillgängliga för att underlätta programmeringen, bland dessa kan nämnas POKE, PEEK, TIN, TOUT och ADDR.

Kompilatorn upptar ca 12k minnesutrymme, medan 'runtime'-rutinerna använder ca 4k. Båda dessa finns med på den kassett som medföljer manualen. 'Runtime'-rutinerna är anpassade för lagring på kassettband.

All kommunikation mellan HP4T och datorn sker genom vektorer, som för bekvämlighetens skull har lagts i början av 'runtimes' (se 'Anvisningar för modifiering av HP4T') - detta gör det enklare för användaren att, om det blir nödvändigt, skriva egna I/O-rutiner.

HP4T använder ett flertal kontrollkoder, vanligen i editorn. Närhelst HP4T väntar att en rad skall läsas in i datorn, kan kontrollkommandona användas på följande sätt:

Kommando	Betydelse	Utförs på SPECTRUM med hjälp av följande tangenter:
RETURN	avslutar raden	<'ENTER'>
CC	återvänder till editorn	<CAPS SHIFT> och <1>
CH	raderar sista tecknet	<CAPS SHIFT> och <0>
CI	går till nästa TAB	<CAPS SHIFT> och <8>
CP	omdirigerar all utskrift från TV-skärmen till printern, eller, om utskrift redan sker på printern, tillbaks till TV-skärmen	fås med hjälp av CHR(16) i en WRITE eller WRITELN-sats
CX	raderar hela raden	<CAPS SHIFT> och <5>
CS	avbryter exekveringen	<CAPS SHIFT> och <SPACE>

Ett enkelt 'loader'-program finns också med så att användaren kan läsa data som lagrats på kassettband i HP4T-format.

HP4T laddas från sida A på det kassettband som medföljer manualen.
Skriv LOAD'' (<K> och två gånger <SYMBOL SHIFT>, <P>)
Tryck ned 'play'-tangenten på kassettbandspelaren, därefter på datorns <ENTER>-tangent. HP4T laddas nu automatiskt. Om 'tape loading error'-meddelandet uppträder på skärmen återspolas bandet och proceduren upprepas, eventuellt efter att ha justerat volymen på kassettbandspelaren.

ZX SPECTRUMs entangentskommandon kan ej användas tillsammans med HP4T (vilket vi ser som ett positivt framsteg). All text måste skrivas in tecken för tecken. <SYMBOL SHIFT> och respektive tangent ger de symboler som är definierade enligt ASCII (utom <I>). Symbolerna < = (<Q>), <> (W) och >=(E) används ej. I stället byggs dessa symboler upp genom kombination av <(<K>), >(<T>) och = (<L>) som hos en vanlig dator.

För att göra en arbetskopia av HP4T går man tillväga på följande sätt:

1. Ladda HP4T som vanligt. Först kommer det ett mycket kort programavsnitt med följande utskrift på TV-skärmen 'Program: HP4S'. När TV-rutan blir tom igen: tryck ned <SHIFT> och <SPACE>-tangenterna samtidigt, varvid laddningen avbryts. Stoppa bandet.

2. Använd entangentskommandon och skriv LOAD 'HP4S' CODE, starta bandet igen och tryck på <ENTER>.

3. Använd ett tomt kassettband för att spara en arbetskopia av HP4S genom följande procedur:
   Skriv SAVE 'HP4S' LINE 1, ställ bandspelaren på inspelning (har Du kommit ihåg att ansluta kabeln till datorn?) och tryck på <ENTER> (därvid sparas 'loadern').

4. När inspelningen är klar: stäng av bandspelaren, skriv SAVE 'HP4S' CODE 24598, 20224. Starta bandspelaren (inspelning) och tryck på <ENTER>.

5. Den kopia som just skapats kan användas på exakt samma sätt som originalbandet.

När HP4T är färdigladdad startar den automatiskt och ger utskriften:

Top of RAM?

Du bör svara med ett positivt heltal (decimalt) som ej är större än 65536 (och därefter <ENTER>), eller bara med <ENTER>. Svarar Du med enbart <ENTER> kommer hela det tillgängliga minnesutrymmet att användas. Genom att svara med ett siffervärde har Du möjlighet att reservera utrymme för egen användning (t.ex. tillägg till kompilatorn). Normalt används endast RAM upp till UDG (se användarmanualen för SPECTRUM).

Nästa fråga är:

Top of RAM for 'T'

Här har Du möjlighet att ändra på startadressen till den stack som skapas med editorns 'T'-kommando (se kapitel 4 för närmare upplysningar). Detta är endast nödvändigt om Du utökat 'runtime'-rutinerna. Normalt svarar Du med <ENTER>.

Den sista frågan lyder:

Table size?

Här har Du möjlighet att avgöra hur mycket utrymme kompilatorns symboltabell får uppta. Svara med ett positivt heltal, eller enbart <ENTER> (varvid 1/16 av det lediga minnesutrymmet tilldelas stacken). I normalfallet räcker det med enbart <ENTER>. Tabellstorleken får aldrig definieras större än 32768.

Om Du lägger till ett 'E' före svaret på den sista frågan kommer inte texteditorn att användas i fortsättningen. Detta medger att man kan skapa sin egen texteditor.

Nu finns kompilatorn och editorn i slutet av symboltabellen, och kontrollen går till editorn.

0.1 Avsikten med denna manual

Denna manual är ej avsedd som en lärobok i Pascal. Har Du ej tidigare erfarenhet av Pascal rekommenderar vi boken 'Pascal from Basic' av Peter Brown (Addison-Wesley, 1982) eller någon annan av de utmärkta böcker som finns i handeln idag.

Denna manual är avsedd som referens till Hisoft Pascal 4.

Kapitel 1 ger syntaxen och semantiken som skall användas till kompilatorn.

Kapitel 2 ger de kommandon och uttryck som finns i Hisoft Pascal 4.

Kapitel 3 innehåller information om de olika tillägg ('options') som kan användas med kompilatorn, samt även formatet på kommentarerna.

Kapitel 4 beskriver editorn, som är en intregrerad del av HP41.

Ovanstående kapitel bör läsas noggrant av alla användare.

Bilaga 1 behandlar de felmeddelanden som ges av kompilatorn och 'runtimes'-rutinerna.

Bilaga 2 består av en förteckning över fördefinierade kommandon och reserverade ord.

Bilaga 3 ger en översikt av den interna datahanteringen i Hisoft Pascal 4.

Bilaga 4 innehåller några prover på Pascal-program. Finner Du några svårigheter med att programmera i Hisoft Pascal 4 kan det ge värdefull hjälp.

0.2 Kompilering och programexekvering

För detaljerad information, se kapitel 4. När kompilatorn aktiveras ger den en utskrift av följande utseende:

```
xxxxnnnn text of source line
```

där xxxx är den adress där koden som genereras börjar och
nnnn är radnumret (utan föregående nollor)

Om en rad är mer än 80 tecken lång sätter kompilatorn automatiskt in ett 'newline' tecken.

Listningen kan styras till en printer med P-option.

Om ett fel upptäcks under kompileringen fås meddelandet '*ERROR*' med en pil som pekar efter den symbol som gav upphov till felet, och ett nummer (se bilaga 1). Det uppstår en paus i listningen. Genom att trycka på 'E' återvänder man till editorn och den rad som gav upphov till felet, 'P' återvänder till editorn och raden som föregick den felaktiga (om sådan finns). Genom att trycka på någon annan tangent fortsätter man kompileringen.

Om kompileringen avslutas på ett felaktigt sätt (t.ex. utan 'END'-kommando) fås utskriften 'No more text' och programmet återgår till editorn.

Om kompilatorn överskrider den angivna tabellstorleken fås utskriften 'No Table Space' och programmet återvänder till editorn. Vanligen lagrar man då programmet på kassett, laddar kompilatorn på nytt och specificerar en större tabell.

Om kompileringen avslutas på ett korrekt sätt men innehåller fel, skrivs antalet ut och objektkoder raderas. Om kompileringen var felfri skrivs frågan 'Run' på skärmen eller printern. Önskas omedelbar exekvering svarar man med 'Y', alla andra svar återför kontrollen till editorn.

Under exekvering av objektkoden kan exekveringsfel ('runtime'-fel) uppstå (se bilaga 1). Du kan avbryta exekveringen genom CS (se sid. 1). CC avslutar exekveringen, medan övriga tangenter fortsätter exekveringen.

0.3 Definition av TYPE

Olika programmeringsspråk har olika sätt att kontrollera huruvida ett dataelement strider mot den definition som tilldelats variabeln eller ej.

Den ena ytterligheten finner vi i maskinkoden, där ingen kontroll ingår i syntaxen. Det är helt upp till programmeraren och användaren att utföra eventuell kontroll.

'Tiny Pascal' enligt 'Byte' tillåter att datatyperna Boolean, Integer och Character blandas fritt utan att ge upphov till felmeddelanden. Detta kan ses som en bit på vägen mot högnivåspråk.

Basic skiljer på teckensträngar och numeriska variabler, ibland även på heltal och flyttal.

Högt upp på skalan kommer Pascal, som tillåter bestämda användardeklarerade typer.

I dagens läge finns ett ännu mer specialiserat språk, nämligen ADA, som tillåter definition av icke-kompatibla numeriska typer bestående av samma slags element.

Det finns två tillvägagångssätt med grundläggande skillnader i olika varianter av Pascal: strukturell ekvivalens och namn-ekvivalens. Hisoft Pascal 4 använder namn-ekvivalens för poster (RECORD) och fält (ARRAYs). Följderna av detta klargöres i kapitel 1. Här nöjer vi oss endast med ett exempel. Låt oss säga att två variabler definieras på följande sätt:

```
VAR A: ARRAY ('A'..'C') OF INTEGER;
    B: ARRAY ('A'..'C') OF INTEGER
```

Man skulle kunna förledas att tro att man kan skriva A:=B; men detta skulle ge upphov till en felutskrift (*ERROR 10*) i Hisoft Pascal 4, eftersom två skilda 'TYPE'-poster har skapats av ovanstående definition. Vill man låta både A och B uttrycka samma slags data måste deklarationen skrivas:

```
VAR A,B: ARRAY ('A'..'C') OF INTEGER;
```

Nu kan användaren fritt tilldela A till B eller omvänt, eftersom endast en 'type'-post har skapats.

I första ögonkastet kan systemet med namnekvivalens verka litet krångligt, men genom att systemet kräver större eftertanke hos programmeraren så undviker man onödiga programmeringsfel.

KAPITEL 1: SYNTAX OCH SEMANTIK

Detta kapitel behandlar syntaxen och semantiken hos Hisoft Pascal 4. Denna följer 'Pascal User Manual and Report' (Jensen/Wirth, 2:a uppl.) där ej annat angivits.

1.1 Identifierare

Endast de 10 första tecknen hos identifieraren anses signifikanta.

Identifieraren får innehålla såväl stora som små bokstäver. Små bokstäver är ej liktydiga med motsvarande stora bokstäver. Så är t.ex. HEJ, HEj och Hej olika. Reserverade ord och fördefinierade identifierare får endast skrivas med stora bokstäver.

1.2 Siffervärden

Heltal har ett absolutvärde av högst 32767 Hisoft Pascal 4. Större tal betraktas som reala.

Mantissan hos reala tal är 23 bits långa. Noggrannheten hos reala tal är ungefär 7 siffror. Noggrannheten går förlorad om resultatet av en beräkning är mycket mindre än absolutvärdena av dess argument. T.ex. ger inte 2.00002-2 resultatet 0.00002. Detta beror på att noggrannheten minskar vid lagring av decimala bråk som binära bråk. Detta fenomen uppstår inte vid användandet av heltal (av acceptabel storlek). 200002-200000 ger t.ex. resultatet 2 (exakt).

Det största reala tal som kan användas är 3.4E38, medan det minsta är 5.9E-39.

Det finns ingen anledning att använda mer än 7 siffror i mantissan eftersom värdet av de överskjutande siffrorna ej beräknas.

När hög noggrannhet eftersträvas bör man undvika inledande nollor eftersom dessa räknas som en siffra. 0.000123456 ger mindre noggrannhet än 1.23456E-4.

Hexadecimala siffror kan användas av programmeraren för att specificera minnesadressen, t.ex. för att länka in maskinkodsavsnitt. Det hexadecimala värdet måste föregås av '#', och minst en hexadecimal siffra måste följa därefter, annars genereras ett '*ERROR*51'.

1.3 Ej tilldelade konstanter

Strängar får ej bestå av mer än 255 tecken. Sträng-'TYPE' är ARRAY (I..N) OF CHAR, där N är ett värde mellan 1 och 255. Textsträngar får ej innehålla EOL (end-of-line)-tecken (CHR(13)). Detta ger upphov till '*ERROR*68'.

Teckenuppsättningen är hela utökade ASCII-koden med 256 tecken. För att bibehålla kompatibiliteten med Standard Pascal får tomt tecken (null character) ej uttryckas '', utan CHR(0) skall användas i stället.

1.4 Konstanten

I Hisoft Pascal 4 finns en konstantdefinition utöver standard, där CHR kan användas för kontrolltecken. Konstanten inom parentesen måste vara av heltalstyp.

```
T.ex.  CONST bs = CHR (10);
             cr = CHR (13);
```

1.5 Enkla typer

Numrerade typer (identifierare, identifierad, ...) får inte ha mer än 256 element.

1.6 TYPE

Det reserverade ordet PACKED godtas, men fyller ingen funktion, eftersom teckensträngar och liknande redan komprimeras. Det enda tillfälle då komprimering vore en fördel är i ett fält av Boolska variabler, men detta uttrycks på ett mer naturligt sätt i annan form.

1.6.1 ARRAY och SET

Den grundläggande typen av SET får ha upp till 256 element. Detta möjliggör SET OF CHAR-definitionen tillsammans med SET av användardefinierad typ. Endast delmängder av heltal i området 0..255 är tillåtna.

Alla kombinationer som ARRAY of ARRAY, ARRAY of SET, RECORD of SET o.s.v. är tillåtna.

Två ARRAY typer behandlas som likvärdiga endast om definitionen härstammar från samma användande av det reserverade ordet ARRAY. Följaktligen är inte nedanstående typer ekvivalenta:

```
TYPE
    tabella = ARRAY (1..100) OF INTEGER;
    tabellb = ARRAY (1..100) OF INTEGER;
```

En variabel av typen tabella får inte tilldelas en variabel av typen tabellb. Detta möjliggör upptäckt av misstag såsom tilldelning av två tabeller som representerar olika data. Ovanstående begränsning gäller ej specialfallet med fält av strängar, eftersom denna typ alltid representerar data av samma typ.

1.6.2 Pekare

Hisoft Pascal 4 tillåter att man skapar dynamiska variabler genom användandet av Standard Pascal-proceduren NEW (se kapitel 2). En dynamisk variabel kan, till skillnad från en statisk variabel (som har minnsutrymme reserverat genom deklarationen), inte nås direkt genom en identifierare eftersom den saknar sådan. I stället använder man en pekarvariabel. Pekarvariabeln är statisk och innehåller adressen till den dynamiska variabeln. Den dynamiska variabeln nås genom att lägga till ett '↑' efter pekarvariabeln. Se exempel i bilaga 4.

Det finns vissa begränsningar i användandet av pekare hos Hisoft Pascal 4. Dessa är som följer:

Pekare till typer som ej deklarerats är inte tillåtna. Detta förhindrar inte att man skapar länkade listor, eftersom typdefinitionen får innehålla pekare till sig själva. T.ex.:

```
TYPE
    item = RECORD
           value: INTEGER;
           next: ↑ item
           END;
    link = ↑ item;
```

Pekare till pekare är inte tillåtna.
Pekare till samma typ anses som likvärdiga. T.ex.:

```
VAR
   first: link;
   current: ↑ item;
```

Variablerna 'first' och 'current' är likvärdiga (d.v.s. man använder strukturell ekvivalens) och får tilldelas varandra eller jämföras.

Den fördefinierade konstanten NIL finns tillgänglig, och när den tilldelas till en pekarvariabel kommer variabeln inte att innehålla någon adress.

1.6.4 RECORD

Poster (RECORD:s) består av ett bestämt antal delar som kallas fält. Till skillnad från Standard Pascal ger inte Hisoft Pascal 4 möjligheten att använda variantdelen av fältet (alltså, CASE går ej att använda).

Två poster anses likvärdiga endast om de deklareras under samma RECORD (se även 1.7.1).

WITH kan användas för att nå olika fält i en post. Detta spar utrymme. Se bilaga 4 för exempel på WITH och RECORD.

1.7 Fältlista

Används tillsammans med RECORD, se 1.7.4 och exempel i bilaga 4.

1.8 Variabel

I Hisoft Pascal finns två sorters variabler, statiska och dynamiska. Statiska variabler deklareras med VAR och minnesutrymme är reserverat för dem under hela programexekveringen.

Dynamiska variabler skapas under programexekveringen av proceduren NEW. De är ej deklarerade och kan ej nås med hjälp av identifierare. De refereras till av en statisk variabel av pekar-typ som innehåller adressen till den dynamiska variabeln. Se kapitel 1.7.2 och kapitel 2 för mer information om dynamiska variabler och bilaga 4 för programexempel.

Vid användande av flerdimensionella fält behöver användaren inte använda samma form på index som i deklarationen. Om en variabel är definierad som ARRAY [1..10] OF ARRAY [1..10] OF INTEGER duger både a[1.][1] och a[1,1] för anrop av elementet [1,1] i fältet.

1.9 Term

Den undre gränsen för ett SET är alltid 0 och storleken är alltid det maximala för den bas som behandlas. Så är t.ex. alltid SET OF CHAR 32 bytes (max 256 element - en 'bit' för varje element). SETO OF 0..10 är lika med SET OF 0..255.

1.10 Enkla uttryck

Samma gäller för enkla uttryck som för termer, se 1.11.

1.11 Uttryck

Vid användande av IN är attributen hela omfånget av det enkla uttrycket förutom heltalsargument, där attributet anses vara (0..255).

Ovanstående syntax gäller när man jämför strängar av samma längd, pekare och alla skalära typer. Jämförelseoperationer som är tillåtna är: >=, <=, <> och =, för pekare dock endast <> och =.

1.12 Parameterlista

Identifieraren för en typ måste följa efter kolon, i annat fall fås *ERROR*44.

OBS!! Procedurer och funktioner är ej giltiga parametrar.

1.13 Påståenden

Vi rekommenderar varmt en titt i syntaxdiagrammet!

Tilldelningar: Se 1.7.

CASE:

en helt tom lista är ej tillåten (*ERROR*13).

ELSE, som är ett alternativ till END i CASE-deklarationen, utförs om väljaren ('expression' i syntaxdiagram) ej är funnen i CASE-listan ('constant' i syntaxdiagram).

Om, i en CASE-sats, END påträffas utan att väljaren funnits fortsätter exekveringen i satsen som följer efter END.

FOR:

Kontrollvariabeln i en FOR-sats får bara vara en ostrukturerad variabel, inte en parameter. Detta är ett mellanting mellan Jensen/Wirth och ISO-standard.

GOTO:

Det är bara möjligt att använda GOTO i samma block som det tilltänkta hoppet och till samma nivå.

Deklaration måste ske med LABEL i det block där det används. En deklaration i en LABEL-sats består av minst en och högst fyra siffror. En 'label' måste stå först i satsen och följas av ett kolon.

FORWARD referens

Procedurer och funktioner får refereras till före deklarationen med hjälp av det reserverade order FORWARD. T.ex.:

```
PROCEDURE a (y:t); FORWARD;
PROCEDURE b (x:y);
    BEGIN
    :
    a (p);
    :
    END;
PROCEDUREa;
    BEGIN
    :
    b (q);
    :
    END;
```

Observera att parametrarna och resultat-typen deklareras tillsammans med FORWARD och inte upprepas i huvuddeklarationen!

1.14 Program

Eftersom filer inte implementeras finns det inte några formella parametrar till ett program.

KAPITEL 2: FÖRDEFINIERADE INDENTIFIERARE

2.1 Konstanter

MAXINT	Det största heltalsvärdet, d.v.s. 32767.
TRUE, FALSE	Booleanska konstanter.

2.2 Typer

INTEGER	Se 1.3.
REAL	Se 1.3 (Flyttal).
CHAR	Hela utökade ASCII-uppsättningen med 256 element.
BOOLEAN	(TRUE, FALSE). Denna typ används i logiska operationer inkluderande resultat i jämförelseoperationer.

2.3 Procedurer och funktioner

2.3.1 Input och Output procedurer

2.3.1.1 WRITE

Proceduren WRITE används för att överföra information till TV-skärmen eller printern. Om uttrycket som skall skrivas är av typen tecken överför WRITE(e) de 8 'bit'-sen som representerar värdet av uttrycket e till skärmen eller printern.

OBS!!
CHR(8) (CTRL H) ger en 'backspace' som kan vara destruktiv.
CHR(12) (CTRL L) raderar skärmen eller ger en ny sida på printern.

CHR(13) (CTRL M) utför radframmatning och flyttar markören till vänster startposition.

CHR(16) (CTRL P) skiftar utskriftsadress från printer till skärm eller tvärtom.

Allmänt:

WRITE (P1, P2, ... Pn); ger samma resultat som:

BEGIN WRITE (P1); WRITE (P2); ...; WRITE (Pn) END;

Parametrarna i en WRITE-sats (P1, P2, e.t.c.) kan ha följande form:

<e> el. <e:m> el. <e:m:n> el. <e:m:H> där e, m och n är uttryck och H är en teckenkonstant.

Det finns 5 olika fall att studera:

1. e är av typen heltal: <e> eller <e:m> kan användas.
Värdet av e omvandlas till en teckensträng med ett mellanslag efter. Längden av strängen kan ökas (med mellanslag) med m som avgör antalet tecken som skall skrivas ut. Om m inte är tillräckligt stort eller saknas skrivs hela e ut med efterföljande mellanslag och m ignoreras.

2. e är av typen heltal och uttrycks i formen <e:m:H>
I detta fall är utskriften hexadecimal. Om m=1 eller m=2 skrivs värdet (e MOD 16↑m) ut med exakt m-tecken. Om m=3 eller m=4 skrivs e ut hexadecimalt i exakt 4 positioner. Om m>4 skrivs hela det hexadecimala värdet ut med utfyllnad av mellanslag vid behov. Utfyllnadsnollor sätts in före värdet där så är befogat. Exempel:

```
WRITE (1025:m:H);
m=1  utskrift: 1
m=2  utskrift: 01
m=3  utskrift: 0401
m=4  utskrift: 0401
m=5  utskrift: _0401
```

3. e är ett flyttal.
Kan presenteras som <e>, <e:m> eller <e:m:n>. Värdet av e omvandlas till en teckensträng. Formatet bestäms av n. Om n inte finns med uttrycks värdet i 'scientific notation', d.v.s. med en mantissa och en exponent. Om värdet är negativt placeras ett minustecken före mantissan, i annat fall föregås talet av ett mellanslag. Talet ges alltid med minst en decimal och högst 5 decimaler. Exponenten föregås alltid av ett tecken, + eller -. Detta innebär att den minsta bredd som kan förekomma vid 'scientific notation' är 8 tecken. Om m<8 ges utskrift med 5 decimaler (12 tecken). Om 8<=m<=12 ges värdet med 1 till 5 decimaler (beroende på m). Om m>12 fylls utskriften av föregående mellanslag. Exempel:

```
WRITE (-1.23E 10:m);
m=7  ger  -1.23000E+10
m=8  ger  -1.2E+10
m=9  ger  -1.23E+10
m=12 ger  -1.23000E+10
m=13 ger  _ -1.23000E+10
```

Om man i stället använder <e:m:n> ges e men n decimaler. Inga mellanslag föregår utskriften om inte m är tillräckligt stort. Om n=0 ges e som heltal. Om e är för stort för att uttryckas i den specificerade formen ges den i stället i 'scientific'-form med bredden m. Exempel:

```
WRITE (IE2:6:2)  ger  100.00
WRITE (IE2:8:2)  ger  _-100.00
WRITE (23.455:6:1)  ger _-23.5
WRITE (23.455:4:2)  ger  -2.34550E+01
WRITE (23.455:4:0)  ger  _-23
```

4. Om e är av typen teckensträng:
antingen <e> eller <e:m> kan användas. Minsta bredd är 1 (enstaka tecken) eller längden av strängen. Utfyllnad med mellanslag framför strängen utförs om m är tillräckligt stor.

5. e är av typen Boolean.
Antingen <e> eller <e:m> kan användas. Utskriften blir 'TRUE' eller 'FALSE' beroende på värdet av e. Minsta bredd 4 eller 5 tecken.

2.3.1.2 WRITELN

WRITELN ger ny rad. Detta är likvärdigt med WRITE (CHR(13)). WRITELN (P1, P2, ... Pn); motsvarar BEGIN WRITE (P1, P2, ... Pn); WRITELN END;

2.3.1.3 PAGE

Proceduren PAGE är likvärdigt med WRITE (CHR(12)); och rensar skärmen eller matar fram printern till början av ett nytt ark.

2.3.1.4 READ

Proceduren READ används för att läsa in data från tangentbordet. Detta sker genom en buffert i 'runtimes'. Från början är detta tomt förutom en EOLN (end-of-line) markering. Vi kan föreställa oss att tillgången till denna buffert sker genom ett textfönster ovanför bufferten som kan se ett tecken i taget. Om textfönstret är placerat över EOLN före en READ-operation är slutförd läses en ny rad text från tangentbordet in.

Under läsning i denna rad går de olika kontrollkoderna (se kapitel 0.0) att använda.

READ (V1, V2 .. Vn); är liktydigt med BEGIN READ (V1); READ (V2);.. READ (Vn) END; där V1, V2 o.s.v. kan betyda tecken, sträng, heltal eller flyttal.

Uttrycket READ (V); har olika verkningssätt beroende på V. Följande 4 fall kan förekomma:

1. V är av typen tecken.
I detta fall läser datorn bara ett tecken från bufferten och tilldelar V motsvarande värde. Om textfönstret befinner sig över EOLN (CHR(13)) kommer funktionen EOLN ge värdet 'TRUE' och en ny textrad läses in från tangentbordet. När en ny READ-operation utförs kommer textfönstret att placeras i början av den nya raden.

OBS! EOLN är 'TRUE' vid programmets början. Om den första READ-operationen är av typ 'character', d.v.s. tecken, kommer CHR(13) bli resultatet. Därefter läses en ny textrad in från tangentbordet. Nästa READ av teckentyp ger första tecknet i raden som svar, om inte raden är blank. Se även READLN nedan.

2. V är av typen sträng.
En teckensträng kan läsas med hjälp av READ, och i detta fall blir resultatet att ett antal tecken läses, antingen tills det sökta antalet tecken har påträffats eller tills EOLN = 'TRUE'. Om EOLN påträffas före det att hela strängen fyllts, fylls resten av strängen av CHR(0). Detta ger programmeraren möjlighet att avgöra längden på den inlästa strängen. Vi vill rikta uppmärksamheten på att samma gäller början av ett program som under 1.

3. V är av typen heltal.
En serie tecken som representerar ett heltal som i 1.3 läses in. Alla föregående blanktecken och EOLN-tecken slopas. Detta innebär att heltal kan läsas omedelbart, jfr 1. ovan.

Om det inlästa har ett absolutvärde större än MAXINT (32767) ges felmeddelandet 'Number too large' och exekveringen avbryts.

Om det första tecknet (förutom EOLN och mellanslag) inte är +, - eller en siffra ges felmeddelandet 'Number expected' och exekveringen avbryts.

4. V är av typen flyttal.
Här läses en serie tecken in enligt syntaxen i 1.3.

Som i 3. ovan måste första tecknet (förutom EOLN och mellanslag) vara +, - eller en siffra. Om talet är för stort eller för litet ges felmeddelandet 'Owerflow'. Påträffas 'E' utan efterföljande tecken eller siffra ges 'Exposant Expected' och påträffas decimalpunkt utan efterföljande siffra ges 'Number Expected'. Flyttal kan, i likhet med heltal, läsas direkt, se 1. och 3. ovan.

2.3.1.5 READLN

READLN (V1, V2, .. Vn); är likvärdigt med: BEGIN READ (V1, V2, .. Vn); READLN END;

READLN läser helt enkelt in en ny buffert från tangentbordet; under tiden som inläsning sker kan de olika kontrollkoderna i kapitel 0.0 användas. EOLN = 'FALSE' efter utfört READLN, om ej den inlästa raden är blank.

READLN kan användas för att förtränga den blanka rad som finns i bufferten vid början av exekveringen, d.v.s. den har effekten att en ny rad läses in i bufferten. Detta är användbart om man vill läsa in en komponent typ enstaka tecken i början av ett program, men är ej nödvändig om man läser in ett hel- eller flyttal.

2.3.2 Input Funktioner

2.3.2.1 EOLN

Denna funktion är Boolsk, och ger värdet 'TRUE' om nästa tecken är CHR(13) (end-of-line), i annat fall värdet 'FALSE'.

2.3.2.2 INCH

Denna funktion avläser tangentbordet och lämnar värdet för den nertryckta tangenten, eller, om ingen tangent är nertryckt, CHR(0). Svaret är av typen tecken ('character').

2.3.3 Överföringsfunktioner

2.3.3.1 TRUNC (X)

Parametern X måste vara av typen hel- eller flyttal och svaret blir ett värde i form av det största heltal som är mindre eller lika med X om X är positivt eller det minsta heltal som är större eller lika med X om X är negativt. Exempel:

TRUNC (-1.5) ger -1
TRUNC (1.9) ger 1

2.3.3.2 ROUND (X)

X måste vara av typ hel- eller flyttal och funktionen ger det 'närmaste' heltalet till X (enligt gängse regler). Exempel:

ROUND (-6.5) ger -6
ROUND (-6.51) ger -7
ROUND (11.7) ger 12
ROUND (23.5) ger 24

2.3.3.3 ENTIER (X)

X måste vara av typen hel- eller flyttal. ENTIER ger det största heltal mindre än eller lika med X. Exempel:

ENTIER (-6.5) ger -7
ENTIER (11.7) ger 11

OBS! ENTIER finns inte i Standard Pascal, men motsvarar BASIC's INT. Det är användbart när man skriver snabba matematiska funktioner. Man förlorar dock i precision (övers. anm.).

2.3.3.4 ORD (X)

X kan vara av valfri typ, dock ej flyttal. ORD ger värdet av ordningstalet för X i den uppsättning i vilken X är definierad.

Om X är av heltalstyp blir ORD (X) = X;, detta bör vanligen undvikas. Exempel:

ORD ('a') ger 97
ORD ('@') ger 64

2.3.3.5 CHR (X)

X måste vara ett heltal. CHR ger ett tecken som i ASCII-koden har värdet av X. Exempel:

CHR (49) ger 'I'
CHR (91) ger ('[')

2.3.4 Aritmetiska funktioner

I samtliga nedanstående exempel måste X vara antingen hel- eller flyttal.

2.3.4.1 ABS (X)

Ger absolutvärdet av X, t.ex. ABS (-4.5) är 4.5. Resultatet är av samma typ som X.

2.3.4.2 SQR (X)

Ger värdet X*X, d.v.s. kvadraten av X. Resultatet är av samma typ som X.

2.3.4.3 SQRT (X)

Ger kvadratroten av X. Svaret är alltid av typen flyttal. Ett 'Math Call Error'-felmeddelande ges om X är negativt.

2.3.4.4 FRAC (X)

Ger decimaldelen av X: FRAC (X) = X - ENTIER (X).

Som med ENTIER gäller det att FRAC är användbart för att skriva snabba matematiska rutiner. Exempel:

FRAC (1.5) ger 0.5 FRAC (-12.56) ger 0.44

2.3.4.5 SIN (X)

Ger sinusvärdet av X där X uttrycks i radianer. Resultatet är av typ flyttal.

2.3.4.6 COS (X)

Ger cosinusvärdet av X där X uttrycks i radianer. Resultatet är av typ flyttal.

2.3.4.7 TAN (X)

Ger tangens av X där X uttrycks i radianer. Svar av flyttalstyp.

2.3.4.8 ARCTAN (X)

Ger vinkeln i radianer, vars tangent är X. Svar av flyttalstyp.

2.3.4.9 EXP (X)

Ger värdet av e↑X där e = 2.71828. Svar av flyttalstyp.

2.3.4.10 LN (X)

Ger den naturliga logaritmen av X. Svar av flyttalstyp. Om x < = 0 ges ett 'Math Call Error'-felmeddelande.

2.3.5 Ytterligare fördefinierade procedurer

2.3.5.1 NEW (p)

Proceduren NEW (p) reserverar plats för en dynamisk variabel. Variabeln p är en pekarvariabel, och efter det att NEW (p) har exekverats innehåller p adressen till den nya dynamiska variabeln. Typen av den dynamiska variabeln är av samma typ som p, och denna kan vara av godtycklig typ. För att få tillgång till den dynamiska variabeln används p, se bilaga 4 för exempel på användning av pekare för att hänvisa till dynamiska variabler. För att återallokera utrymme som använts för dynamiska variabler används procedurerna MARK och RELEASE (se nedan).

2.3.5.2 MARK (v1)

Denna procedur sparar de upprättade dynamiska variablerna i pekarvariabeln v1. För att återställa de dynamiska variablerna till den status de hade när MARK exekverades använder man proceduren RELEASE (se nedan).

2.3.5.3 RELEASE (v1)

Denna procedur frigör minnesutrymmet i stacken för användande av dynamiska variabler. Stacken återställs i det läge (status) som den hade när MARK exekverades. Därvid förstörs alla dynamiska variabler som skapats efter exekveringen av MARK.

Vi rekommenderar försiktighet och eftertanke vid användandet av RELEASE, eftersom de frigjorda minnespositionerna inte kan återskapas, och all information i dessa ohjälpligt går förlorad.

2.3.5.4 INLINE (C1, C2, C3, ...)

Denna procedur tillåter insättande av Z80 maskinkod i Pascalprogrammet. Värdena (C1, C2, C3...) sätts in i objektprogrammet i den aktuella minnesadressen hos kompilatorn. C1, C2, C3 är heltal (OBS! i området 0..255 (decimalt) = 00- FF (hexadecimalt)). Antalet värden i en INLINE-sats är ej begränsat. Se bilaga 4 för programexempel med INLINE.

2.3.5.5 USER (V)

USER är en procedur med ett heltalsargument V. Proceduren anropar minnesadressen V. Eftersom Hisoft Pascal 4 lagrar heltal i tvåkomplement-form (se bilaga 3) måste man, vid anrop av minnesadresser större än 7FFF (32767), använda negativa värden på V. För att t.ex. anropa C000 används -16384 och USER (-16384) ger ett hopp till den önskade adressen. För bekvämlighets skull rekommenderar vi anrop med hexadecimala argument.

Den anropade rutinen skall avslutas med en Z80 RET-instruktion (C9) och måste spara IX-registret (d.v.s. IX-registret skall vid återhopp ha samma värde som det hade vid anropet av rutinen. övers. anm.)

2.3.5.6 HALT

Denna procedur förorsakar ett stopp i exekveringen och ger meddelandet 'Halt at PC = XXXX' där XXXX ger den hexadecimala adressen där HALT påträffades. Tillsammans med en kompileringslistning kan HALT användas för att avgöra vilken av två eller flera möjliga vägar programexekveringen tar. Detta används vanligen vid felsökning.

2.3.5.7 POKE (X, V)

POKE lagrar uttrycket V i datorns minne med början i adressen X. X är ett heltal och V är av valfri typ, dock ej SET. Se 2.3.5.5 angående representation av minnesadresser. Exempel:

POKE (6000, 'A') lägger ut 41 i minnesadress 6000.
POKE (-16384, 3.6E3) lägger ut 00 0B 80 70 (hexadecimalt) i minnesadress C000.

2.3.5.8 TOUT (NAME, START, SIZE)

Proceduren TOUT används för att lagra variabler på kassettband. Den första parametern är av typ ARRAY 1..8 OF CHAR, och utgör namnet på den fil som skall sparas. SIZE ger det antal 'bytes' som skall sparas med början från minnesadress START. Dessa två parametrar är av typ heltal.

T.ex. för att spara variabeln V på kassettband under namnet 'VAR':

TOUT ('VAR', ADDR (V), SIZE (V))

Användandet av verkliga minnesadresser ger användaren större möjligheter än om man bara kunde spara fält. T.ex. kan en hel skärm sparas direkt. Se bilaga 4 för exempel på TOUT.

2.3.5.9 TIN (NAME, START)

Denna procedur används för att läsa in från kassettband, t.ex. variabler som lagrats med TOUT. NAME är av typ ARRAY 1..8 OF CHAR och START är av typ heltal. Bandet avsöks efter en fil med namnet NAME som läses in med början i minnesadress START. Antalet 'bytes' som läses in tas från bandet, eftersom det redan finns undanlagrat med TOUT:

För att t.ex. läsa in variabeln från exemplet i 2.3.5.8 gör man så här:

TIN ('VAR', ADDR (V))

Eftersom källfilen är inspelad med hjälp av editorn i samma format som TIN och TOUT kan TIN användas för att läsa in textfiler till ARRAY OF CHAR för behandling.

Se bilaga 4 för exempel på TIN.

2.3.5.10 OUT (P, C)

Denna procedur används för att nå Z80-processorns utportar utan att använda INLINE. Värdet av heltalsparametern P lagras i BC-registret, teckenparametern C lagras i A-registret och assemblerinstruktionen OUT (C), A utförs. Exempel:

OUT (1, 'A') lägger ut tecknet 'A' till Z80 port 1.

2.3.6 Fler fördefinierade funktioner

2.3.6.1 RANDOM

Detta ger ett pseudo-slumptal mellan 0 och 255. Denna rutin ger den dåligt resultat i slingor som inte innehåller I/O operationer.

Om användaren behöver en bättre rutin, rekommenderar vi att man skriver en ny, antingen i Pascal eller i maskinkod, som bättre motsvarar de speciella krav som finns för applikationen.

2.3.6.2 SUCC (X)

X får vara av valfri typ, dock ej flyttal. SUCC (X) ger efterföljande värde till X. Exempel:

SUCC ('A') ger 'B' SUCC (5') ger '6'

2.3.6.3 PRED (X)

X som ovan. PRED (X) ger föregående värde till X. Exempel:

PRED ('j') ger 'i' PRED (TRUE) ger FALSE

2.3.6.4 ODD (X)

X skall vara heltal. ODD ger ett Boolskt värde som är TRUE om X är udda och FALSE om X är jämnt.

2.3.6.6 ADDR (V)

Denna funktion tar en identifierare av valfri typ som parameter och ger ett heltal som är minnesadressen av identifieraren V. För närmare upplysning om hur variabler lagras (under runtime) i Hisoft Pascal 4, se bilaga 4. För exempel på användande av ADDR se bilaga 4.

2.3.6.7 PEEK (X, T)

Första parametern i funktionen är av typ heltal och avser den minnesadress som skall avläsas. Den andra parametern anger hur resultatet skall tolkas.

PEEK används för att läsa data direkt från en given minnesposition. Resultatet kan bli av vilken typ som helst.

I alla PEEK och POKE-operationer flyttas data i enlighet med Hisoft Pascal 4:s interna representation som beskrivs närmare i bilaga 3. Om vi antar att minnet från 5000 (hexadecimalt) innehåller följande värden: 50, 61, 73, 63, 61, 6C (hexadecimalt) får vi följande resultat i exemplen nedan:

```
WRITE (PEEK ( 5000, ARRAY (1..6) OF CHAR)) ger 'Pascal'
WRITE (PEEK ( 5000, INTEGER)) ger 24912
WRITE (PEEK ( 5000, REAL)) ger 2.46227E+29
```

2.3.6.7 SIZE (V)

Parametern i denna funktion är en variabel. Resultatet är ett heltal som anger hur mycket utrymme (i 'bytes') variabeln upptar.

2.3.6.8 INP (P)

INP används för att nå Z80-portarna utan att använda INLINE. Värdet av heltalsparametern P lagras i BC-registret och resultatet, ett tecken, fås genom exekveringen av assemblerinstruktionen IN A, (C).

KAPITEL 3: KOMMENTARER OCH KOMPILATOR-'OPTIONS'

3.1 Kommentarer

Kommentarer kan läggas mellan två reserverade ord, siffror, identifierare eller speciella symboler, se bilaga 2. En kommentar börjar med '{' eller '(*'. Om nästa inte är '$' ignoreras alla tecken tills nästa '}' eller '*)' påträffas. Om '$' påträffas söker kompilatorn efter en serie kompilator-'options' (se nedan). Därefter ignoreras alla tecken tills '}' eller '*)' påträffas.

3.2 Kompilator-'options'

Följande 'options' är tillgängliga:

Option L: Kontrollerar listningen av programtext och objektkod-adresser i kompilatorn. Om L+ ges en fullständig listning.
Om L- ges endast listning av raderna om fel påträffas.
Normalt: L+.

Option O: Kontrollerar om vissa 'owerflow'-kontroller skall utföras eller ej. Heltalsmultiplikation och division och alla flyttalsoperationer kontrolleras alltid.
Om O+ kontrolleras även heltalsaddition och subtraktion.
Om O- görs ej denna kontroll.
Normalt: O+.

Option C: Styr huruvida tangentbordet skall kontrolleras eller ej under programexekveringen.
C+ medför att tangenten CC ger ett HALT-meddelande och ett avbrott i exekveringen.
Kontrollen görs i början av alla slingor, procedurer och funktioner. Detta kan användas för att kontrollera vilken loop som inte avslutas på ett korrekt sätt vid felsökning. Genom att avstå från denna kontroll ökas exekveringshastigheten avsevärt.
C- medför att kontrollen ej utföres.
Normalt: C+.

Option S: Avgör om kontroll av stacken utförs.
Om S+ kontrolleras stacken före varje procedur- och funktionsanrop med avseende på om stacken löper risk att överskridas i detta block. Om 'runtime'-stacken börjar skriva i de dynamiska variablerna eller i programmet fås meddelandet 'Out och RAM at PC=XXXX' och exekveringen avbryts. Självklart är detta inte helt säkert. Om en procedur använder en mycket stor area av stacken kan programmet 'krascha'. På motsvarande sätt kan exekveringen avbrytas i onödan om en procedur använder ovanligt lite av stacken.
Om S- kontrolleras inte stacken.
Normalt: S+.

Option I: Vid användande av 16 'bit' tvåkomplement heltalsaritmetik sker 'owerflow' när operationerna >, <, >= eller <= utföres och skillnaden är större än MAXINT (32767). När detta sker blir resultatet av jämförelsen felaktig. Normalt inträffar ej detta fenomen, men om användaren skulle vilja jämföra sådana tal medför användandet av I+ att resultatet blir korrekt. Motsvarande situation kan uppstå vid jämförelse av flyttal som har en större skillnad än 3.4E38. Detta kan dock inte undvikas.
Om I- utföres ingen kontroll vid jämförelseoperationer.
Normalt: I-.

Option P: Om P-option används ändras utskriften från TV-skärmen till printern eller tvärtom. Denna option följs inte av + eller -.
Normalt: TV-skärmen används.

KAPITEL 4: DEN INBYGGDA EDITORN

4.1 Introduktion av editorn

Editorn är en enkel radbaserad editor skapad med avseende på enkel, snabb och effektiv användning. Text lagras i minnet i komprimerad form. Antalet mellanslag före varje rad lagras som en teckenposition i början av raden och alla HP4T Reserverade ord lagras som ett tecken. Detta ger en komprimering av texten av ca 25%.

Editorn startar automatiskt när HP4T laddas och ger utskriften:

och markören '>'

Användaren kan nu svara med en kommandorad med följande format:

C N1, N2, S1, S2

därefter ENTER, där koderna har följande betydelse:

C	är det kommando som skall utföras (se kapitel 4.2 nedan)
N1	är ett tal mellan 1 och 32767
N2	är ett tal mellan 1 och 32767
S1	är en teckensträng med max 20 tecken
S2	är en teckensträng med max 20 tecken

Kommatecknet används för att skilja argumenten åt (detta kan dock ändras - se 'S'-kommandot), mellanslag ignoreras utom i strängarna. Inget av argumenten är obligatoriskt. Vissa kommandon exekveras dock inte utan N1 och N2 (t.ex. Delete-kommando). Editorn minns de argument som givits i tidigare kommandon och använder dessa där så är lämpligt, om inte nya argument ges. N1 och N2 är från början 10 och strängarna tomma. Om ett felaktigt kommande ges, t.ex. F-1, 100, HEJ, ignoreras detta och texten 'Pardon?' skrivs på skärmen. Skriv in det nya, korrekta kommandot, t.ex. F1, 100, HEJ. Felmeddelandet ges också om S2 är längre än 20 tecken. Om S1 är för långt slopas överskjutande tecken.

Kommandon kan ges med stora eller små bokstäver.

När en kommandorad skrivs kan alla relevanta kontrollfunktioner från kapitel 0.0 användas.

Följande stycke behandlar de olika kommandona i editorn. Observera att om ett argument är omgivet av <> måste det finnas med för att kommandot skall utföras.

4.2 Editorkommandon

4.2.1 Tillägg av text

Text kan läggas till en textfil antingen genom att skriva ett radnummer, ett mellanslag och den önskade texten eller med 'I'-kommandot. Observera att om ett radnummer följs av 'ENTER' (d.v.s. utan någon text) raderas hela raden och all text som tidigare fanns går förlorad. När text skrivs in kan SC (radera början av raden), CI (nästa tab-position), CC (återgå till Kommando-slingan) och CP (printer TV el vice versa) användas.

DELETE ger en destruktiv radning, men endast i den rad som editeras. Den text som skrivs lagras i en intern buffer, och skulle denna fyllas måste DELETE eller CX användas.

Kommando: I n, m — Ger möjlighet till insättning, med början på rad n och med steglängden m. Texten skrivs in och raden avslutas med ENTER. För att gå in i I-mode används CC.

Om användaren skriver text på ett radnummer som redan existerar förstörs tidigare text sedan ENTER använts. I-kommandot avbryts om radnummer större än 32767 genereras.

Om den skrivna texten är längre än radlängden på skärmen görs en automatisk scroll och tabulering, så att radnummer avskiljs klart, varefter utskriften kan fortsätta. Radlängden begränsas dock av storleken på buffern, som är 128 tecken lång.

4.2.2 Listning av text

Text kan listas med 'L'-kommandot. Antalet rader är fördefinierat, men kan ändras med 'K'-kommando.

Kommando: L n,m — Listar texten från rad n t.o.m. m. Om n och m ej anges antas de alltid vara 1 respektive 32767. För att scrolla texten kan man trycka på valfri tangent, dock ej CC som avbryter listningen.

Kommando: K n — 'K' ger antal rader som visas samtidigt på TV-skärmen. T.ex. K5 ger 5 rader åt gången när 'K'-kommandot används.

4.2.3 Textediterіng

Kommandon finns för att lägga till, radera, flytta och omnumrera rader.

Kommando: D<n,m> — Alla rader från n:r n till och med m raderas. Om m < n sker ingenting. En enskild rad kan raderas genom att sätta m = n eller genom att skriva radnumret och sedan göra ENTER.

Kommando: M n,m — Medför att radnummer n kopieras till radnummer m, varvid all tidigare text på m förstörs. Observera att rad n finns kvar. Om en rad med nummer n ej existerar händer ingenting.

Kommando: N<n,m > Omnumrerar alla rader med första nummer n och steglängd m. Överskrids 32767 bibehålls den gamla numreringen. Både n och m måste alltid anges.

Kommando: F n,m,f,s Texten mellan radnummer n<x<m genomsöks med avseende på strängen f. Hittas denna text visas raden och kontrollen övergår till editorn (se nedan). Användaren kan nu söka för ytterligare förekomst av f eller ersätta f med strängen s (se nedan). n och m kan ha definierats tidigare, så att endast 'F' behöver användas (se kapitel 4.3).

Kommando: E n Editerar rad n. Om n ej existerar sker ingenting, i annat fall kopieras raden till en buffert och visas på skärmen. En kopia av radnumret visas också. Därvid kan editering ske. All editering sker i bufferten, så att den ursprungliga raden skall kunna nås vid behov. Vid editering kan följande kommandon användas. (En pekare tänks röra sig genom raden med början i första tecknet.)

' ' (Space) - flyttar pekaren ett steg framåt. Man kan inte gå förbi radslutet.

DELETE flyttar pekaren ett steg bakåt (!). Medger ej att man går förbi första tecknet.

CI (kontrollfunktion) - flyttar pekaren till nästa tap-position.

ENTER avslutar editeringen och verkställer alla ändringar.

Q - avslutar editeringen utan att verkställa de utförda ändringarna. Raden behålles i ursprungligt skick.

R - laddar bufferten med originalraden och ignorerar alla hittills gjorda ändringar.

L - skriver ut resten av raden efter pekarpositionen och flyttar markören till första tecknet, fortfarande i Edit-mode.

K- raderar det tecken som markören pekar på.

Z - raderar det tecken som markören pekar på och alla efterföljande tecken i raden.

F - söker efter nästa förekomst av f-strängen (se F-kommandot). Detta kommando går ut ur Edit-mode och verkställer alla ändringar om inte en ny f-sträng påträffas i samma rad.

S - ersätter den funna f-strängen (se F-kommandot) med s-strängen och fortsätter sökningen enligt F-kommandot. Se exempel i kapitel 4.3.

I - tillåter att ny text skrivs in i den aktuella markörpositionen. Genom ENTER återgår man till Edit-mode och markören ställer sig efter det sista skrivna tecknet. DELETE ger i detta fall till resultat att tecknet närmast till vänster om markören raderas. CI (kontrollfunktion) flyttar markören till nästa tab-position och fyller ut mellanrummet med blanktecken.

X - flyttar markören till radslutet och övergår automatiskt i I-mode.

C - tillåter att det tecken som markören pekar på skrivs över och därvid flyttas markören till nästa position. C-mode bibehålles tills ENTER används, varvid Edit-mode återfås. Markören pekar på det tecken som senast ändrades. DELETE flyttar markören ett steg bakåt.

4.2.4 Bandspelarkommandon

Text kan sparas med 'P' och 'G' kommandon på kassettband.

Kommando: P n,m,s — Raderna n<x<m sparas på kassettband i HP4T-format under filnamnet s. Kom i håg att n, m och s kan ha definierats av tidigare kommandon, och bör därför skrivas om för säkerhets skull. Försäkra dig om att bandspelaren är påslagen för inspelning före användandet av detta kommando. Medan texten spelas in visas texten 'Busy ...'.

Kommando: G,,s — Bandet avsöks efter en fil i HP4T-format med namnet s. Under tiden visas texten 'Busy ...'. Om en fil med felaktigt filnamn påträffas ges utskriften 'Found' + filnamn och sökandet fortsätter. Påträffas en fil med korrekt filnamn ges utskriften 'FOUND' och filen läses in. Om ett fel påträffas under inläsning ges utskriften 'Checksum error', varvid inläsningen avbryts.

Om strängen s är tom laddas första bästa HP4T-fil som påträffas. Medan sökning pågår kan denna avbrytas genom att CC trycks in.

Observera att om en textfil redan finns i minnet kommer den nya filen att läsas in efter den redan existerande och hela texten omnumreras med första radnummer 1 och steglängden 1.

4.2.5 Kompilering och exekvering från editorn

Kommando: C n: — Kompilerar texten från och med radnummer n. Om n inte anges kompileras hela texten. Se även kapitel 0.2.

Kommando: R: — Den tidigare kompilerade objektskoden exekveras, men endast om källkoden inte utökats under tiden. Se även kapitel 0.2.

Kommando: T n:

Detta är ett översättningskommando. Den existerande källkoden kompileras fr.o.m. radnummer n (eller från början om n utelämnas) och, om kompileringen är felfri, ger utskriften 'OK?'. Genom svaret 'Y' förflyttas objektkoden till slutet av 'runtimes' (därvid förstörs kompilatorn!) och 'runtimes' tillsammans med objektkoden sparas på kassettband under filnamnet som definierades under 'f'-strängen. Nu kan man ladda in filen i datorns minne med LOAD'', varvid exekveringen påbörjas automatiskt.

Observera att man är tvungen att ladda kompilatorn på nytt efter 'T'-kommandot, men detta utgör inget stort problem, eftersom 'T'-kommandot endast används när ett program fungerar tillfredsställande och skall lagras för vidare användning.

Om någon annan tangent än 'Y' används som svar på frågan 'OK?' återgår kontrollen till editorn.

4.2.6 Övriga kommandon

Kommando: B

Återför kontrollen till operativsystemet. För att återgå till HP4T används GOTO 9 (varvid källtexten sparas) eller GOTO 12 (varvid källtexten raderas).

Kommando: O n,m

Kan användas för att komprimera text som inte skrivits in under den vanliga editorn. Denna operation tar ganska lång tid att utföra. n och m anger radnummer som ovan.

Kommando: S,,d

Detta kommando tillåter ändring av det tecken som används för att separera argumenten till kommandon. I normala fall används ',' (komma), detta kan ändras till det första tecknet i strängen d genom 'S'-kommandot.

Kom i håg att hädanefter måste alltid det nya tecknet användas tills ett nytt är specificerat, även i 'S'-kommando. Som avgränsning får alla tecken utom blanktecken användas.

4.3 Exempel på användande av editorn

Låt oss anta att följande program har skrivits in (med hjälp av I 10,10):

```
10   PROGRAM BUBBLESORT
20   CONST
30     Size = 2000;
40   VAR
50     Numbers: ARRAY [1..Size] OF INTEGER;
60     I, Temp: INTEGER;
70   BEGIN
80     FOR I: = 1 TO Size DO Number [I]: = RANDOM;
90     REPEAT
100      FOR I: = 1 TO Size DO
110      Noswaps: = TRUE;
120      IF Number [I] > Number [I+1] THEN
130        BEGIN
140          Temp: = Number [I];
150          Number [I]: = Number [I + 1];
160          Number [I + 1]: = Temp;
170          Noswaps: = FALSE
180        END
190    UNTIL Noswapss
200  END.
```

Detta program innehåller följande fel:

Rad 10	Semikolon saknas
Rad 30	Inget egentligt fel, men ändra ändå till 100
Rad 100	'Size' ändras till 'Size - 1'
Rad 110	Denna rad skulle ha radnummer 95 i stället
Rad 190	'Noswapss' skall stavas 'Noswaps'

Dessutom har variabeln 'Numbers' blivit deklarerad men refereras till som 'Number'. Slutligen har den Boolska variabeln ej deklarerats.

Försök först rätta till felen själv med hjälp av editorn. Lösningen följer nedan:

F 60, 200, Number, Numbers	och använd 'S'-kommandot nödvändigt antal gånger
E 10	därefter: X; ENTER ENTER
E 30	därefter ..... K C 1 ENTER ENTER
F 100, 100, Size, Size - 1	därefter 'S'
M 110, 95	
110	därefter ENTER
E 190	därefter X DELETE ENTER ENTER
65 Noswaps: BOOLEAN;	
N 10, 10	numrera om

Vi rekommenderar varmt att användaren arbetar sig igenom ovanstående exempel, inte bara teoretiskt utan även genom att verkligen använda editorn. För den som inte använt texteditor av liknande sort kan det först verka en smula besvärligt, men man vänjer sig snabbt.

BILAGA 1: FELUTSKRIFTER

A.1.1 Felnummer som ges av kompilatorn

1. För stort tal
2. Semikolon förväntas
3. Ej deklarerad identifierare
4. Använd '=' i stället för ':=' i konstantdeklaration
6. '=' förväntas
7. Denna identifierare kan inte påbörja en sats
8. ':=' förväntas
9. ')' förväntas
10. Fel typ
11. '.' förväntas
12. Faktor förväntas
13. Konstant förväntas
14. Denna identifierare är ej en konstant
15. 'THEN' förväntas
16. 'DO' förväntas
17. 'TO' eller 'DOWNTO' förväntas
18. '(' förväntas
19. Kan ej skriva denna typ av uttryck
20. 'OF' förväntas
21. ',' förväntas
22. ':' förväntas
23. 'PROGRAM' förväntas
24. Variabel förväntas eftersom parametern är en variabel
25. 'BEGIN' förväntas
26. Variabel förväntas som svar på READ
27. Kan ej jämföra uttryck av denna typ
28. Skall vara antingen INTEGER eller REAL
29. Kan ej läsa denna typ av variabel
30. Denna identifierare är ej en typ
31. Exponent förväntas i flyttal
32. Ej numeriskt uttryck förväntas
33. Nollsträng ej tillåten (använd CHR(0))
34. '[' förväntas
35. ']' förväntas
36. Fältindex måste vara numerisk
37. '..' förväntas
38. ']' eller ',' förväntas i ARRAY-deklaration
39. Undre gräns större än övre gräns
40. För stor uppsättning (fler än 256 element)
41. Resultatet av funktionen måste vara av typ identifierare
42. ';' eller ']' förväntas
43. '..' eller ',' eller ']' förväntas
44. Parametern måste vara av identifierartyp
45. Tom uppsättning kan inte vara första faktor i en icke-tilldelningssats
46. Numeriskt (inkl flyttal) förväntas
47. Numeriskt (exkl flyttal) förväntas
48. Uppsättningarna ej kompatibla
49. '<' och '>' kan ej användas vid jämförelser av uppsättningar
50. 'FORWARD', 'LABEL', 'CONST', 'VAR', 'TYPE' eller 'BEGIN' förväntas
51. Hexadecimalt tal förväntas
52. Kan ej POKE:a uppsättningar
53. Fält för stort (>64k)
54. 'END' eller ';' förväntas i RECORD definition
55. Fältidentifierare förväntas

56. Variabel förväntas efter 'WITH'
57. Variabel i 'WITH' måste vara av RECORD-typ
58. Fältidentifierare ej associerad med WITH-uttryck
59. Heltal utan tecken förväntas efter 'LABEL'
60. Heltal utan tecken förväntas efter 'GOTO'
61. Denna 'label' är på fel nivå
62. Ej deklarerad 'label'
63. Parametern till SIZE förväntas vara en variabel
64. Kan endast testa för likhet hos pekare
67. Enda utskriftsparameter för heltal med två ':' är e:m:H
68. Strängar får ej innehålla EOL-tecken
69. Parameter till NEW, MARK eller RELEASE skall vara av typen variabel eller pekare
70. Parametern till ADDR förväntas vara en variabel

A.1.2 Runtime felmeddelanden

När ett fel i 'runtime' upptäcks ges en av följande felmeddelanden följt av 'at PC = XXXX' där XXXX är minnesadressen till felet. Oftast är felet uppenbart. Om så ej är fallet; jämför med kompileringslistan.

1. Halt
2. 'Owerflow'
3. Out of RAM
4. Division med noll (genereras också av DIV.)
5. Index för litet
6. Index för stort
7. Maths Call Error
8. För stort tal
9. Siffra förväntas
10. För lång sträng
11. Exponent förväntas

Runtime-fel ger avbrott i exekveringen.

BILAGA 2: RESERVERADE ORD OCH FÖRDEFINIERADE IDENTIFIERARE

A.2.1 Reserverade ord

AND	ARRAY	BEGIN	CASE	CONST	DIV	DO
DOWNTO	ELSE	END	FORWARD	FUNCTION	GOTO	IF
IN	LABEL	MOD	NIL	NOT	OF	OR
PACKED	PROCEDURE	PROGRAM	RECORD	REPEAT	SET	THEN
TO	TYPE	UNTIL	VAR	WHILE	WITH	

A.2.2 Specialsymboler

+	-	*	/		
=	<>	<	<=	>=	>
(	)	[	]		
#	}		(*	*)	
↑	:=	.	,	;	:
'	..				

A.2.3 Fördefinerade identifierare

Nedanstående enheter anses deklarerade i ett block omgivande hela programmet och finns därför tillgängliga i hela programmet om de ej omdeklareras i ett av de inre blocken. Se även kapitel 2.

```
CONST       MAXINT = 32767;

TYPE        BOOLEAN = (FALSE, TRUE);
                 CHAR (Hela utökade ASCII-uppsättningen);
                 INTEGER = -MAXINT..MAXINT;
                 REAL (Flyttal. Se kapitel 1.3)

PROCEDURE   WRITE; WRITELN; READ; READLN; PAGE; HALT; USER; POKE; INLINE;
            OUT; NEW; MARK; RELEASE; TIN; TOUT;

FUNCTIONS   ABS; SQR; ODD; RANDOM; ORD; SUCC; PRED; INCH; EOLN; PEEK; CHR;
            SQRT; ENTIER; ROUND; TRUNC; FRAC; SIN; COS; TAN; ARCTAN; EXP;
            LN; ADDR; SIZE; INP;
```

BILAGA 3: DATAREPRESENTATION OCH LAGRING

A.3.1 Datarepresentation

Följande behandlar hur data representeras internt av Hisoft Pascal 4.

Informationer angående det utrymme som varje operation tar i minent är till nytta för de flesta programmerare. övriga detaljer kan vara användbara för dem som önskar blanda Pascal och maskinkod.

A.3.1 Heltal

Heltal upptar 2 byte var, i tvåkomplement. Exempel:

```
1     =    0001
256   =    0100
-256  =    FF00
```

Vanligen använder kompilatorn Z80's HL-register för heltal.

A.3.1.2 Tecken, Boolean och andra skalära enheter

Dessa upptar 1 byte var i vanlig binär form.
Tecken: 8 'bit', extended ASCII används
'E' =#45
'[' =#5B

Boolean:

ORD (TRUE) = 1	TRUE representeras av 1
ORD (FALSE) = 0	FALSE representeras av 0

Vanligen används Z80's A-register.

A.3.1.3 Flyttal

(Mantissa, Exponent)-form används, ungefär som i vanlig Scientific Notation. Binärkod används dock i stället för decimalkod. Exempel:

$2 = 2 * 10^0$ eller $1.0_2 * 2^1$

$1 = 1 * 10^0$ eller $1.0_2 * 2^0$

$-12{,}5 = -1.25 \times 10^1$ eller $-25 * 2^{-1} =$

$= - 11001_2 * 2^{-1} =$

$= - 1.1001_2 * 2^3$

$0{,}1 = 1.0 * 10^{-1}$ eller $\frac{1}{10} = \frac{1}{1010_2} = \frac{0.1_2}{101_2}$

binär division ger:

```
          0.0001100
     ____________
101 | 0.1100000000
```

härefter återupprepar sig funktionen.

$$= \frac{0.1_2}{101_2} = 0.0001100_2$$

Svaret blir $1.1001100 * 2^{-4}$

Hur används då detta för att lagra data i datorn? Först och främst reserveras 4 bytes för varje flyttal i följande format:

tecken	normaliserad mantissa	exponent	data
23	22 0	7 0	'bit'
H	L	E D	register

tecken: mantissans tecken: 1 = negativ, 0 = positiv

normaliserad mantissa: normaliserad till formen 1.xxxxxx med högsta 'bit:en' ('bit' 22) alltid 1, om ej 0 skall uttryckas (HL = 0, DE = 0)

exponent: exponent i binär tvåkomplementform

OBS! Flyttal är lagrade i minnet i formen EDLH (Intel representation. Övers.anm.)

A.3.1.4 Poster och fält

Poster använder minnesutrymme som motsvarar summan av komponenternas minnesutrymme.

Fält: om n = antal element och s = storleken på varje element krävs n * s minnespositioner

t.ex. behöver ARRAY [1..10] OF INTEGER 10 * 2 = 20 bytes.

ARRAY [2..12, 1..10] OF CHAR har 11 * 10 = 110 element och behöver därför 110 byte minnsutrymme.

A.3.1.5 Uppsättningen ('sets')

Uppsättningen lagras som 'bit'-strängar, så att om grundtypen har n element behövs ((n-1) DIV 8+1) bytes. Exempel:

SET OF CHAR behöver (256-1) DIV 8+1 = 32 bytes.
SET OF (blue, green, yellow) behöver (3-1) DIV 8+1 = 1 byte.

A.3.1.6 Pekare

Pekare upptar 2 bytes som innehåller adressen (I Intel-format, lägsta byte först) till den variabel pekaren pekar på.

A.3.2 Variabellagring under 'Runtime'

Det finns 3 tillfällen när användaren behöver känna till lagringen under 'Runtime':

1. Globala variabler - deklarerade i huvudblocket
2. Lokala variabler - deklarerade i något inre block
3. Parametrar och svarsvärden - överförs mellan procedurer och funktioner och huvudprogram.

Dessa fall tas upp nedan och exempel på praktisk tillämpning står att finna i bilaga 4.

Globala variabler

Globala variabler lagras i 'Runtimes' stack uppifrån och ned. Om t.ex. 'Runtimes' stack börjar i #B000 och huvudprogrammets variabler är:

```
VAR   i: INTEGER;
     ch: CHAR;
      x: REAL;
```

så finns i, som upptar 2 bytes i #B000 - 2 och #B000 - 1, d.v.s. #AFFE och #AFFF. ch tar 1 byte och finns i #AFFE - 1 = #AFFD. x tar 4 byte och finns i #AFF9 - #AFFC.

Lokala variabler

Lokala variabler är inte lika lätta att tillgå från stacken. Vid början av varje block sätts IX så att IX - 4 pekar på början av blockets lokala variabler. Exempel:

```
PROCEDURE   test;
VAR         i, j: INTEGER
```

i tar 2 byte och finns i IX - 4 - 2, d.v.s. IX - 6 och IX - 5.
j finns i IX - 8 och IX - 7.

Parametrar och svarsvärden

Värdeparametrar behandlas som lokala variabler, och likt dessa, ju tidigare den är deklarerad ju högre adress har den. I motsats till lokala variabler är den lägsta (inte den högsta) adressen fixerad i IX + 2. Exempel:

```
PROCEDURE   test (i:REAL; j:INTEGER);
```

j (först allokerad) finns i IX + 2 och IX + 3.
i finns i IX + 4 IX + 7.

Variabler behandlas som värdeparametrar, med det undantaget att dessa alltid allokerar 2 bytes som innehåller adressen till variabeln. Exempel:

```
PROCEDURE   test (i:INTEGER; x:REAL);
```

referensen till x placeras i IX + 2 och IX + 3. Dessa positioner anger adressen där x är belägen.
Värdet av i finns i IX + 4 och IX + 5.

Svarsvärden från funktioner placeras alltid i första parametern i minnt. Exempel:

```
FUNCTION  test (i:INTEGER): REAL;
```

i finns i IX + 2 och Ix + 3 och minne finns reserverat för svaret i IX + 4, IX + 5, IX + 6 och IX + 7.

BILAGA 4: NAGRA PROGRAMMERINGSEXEMPEL

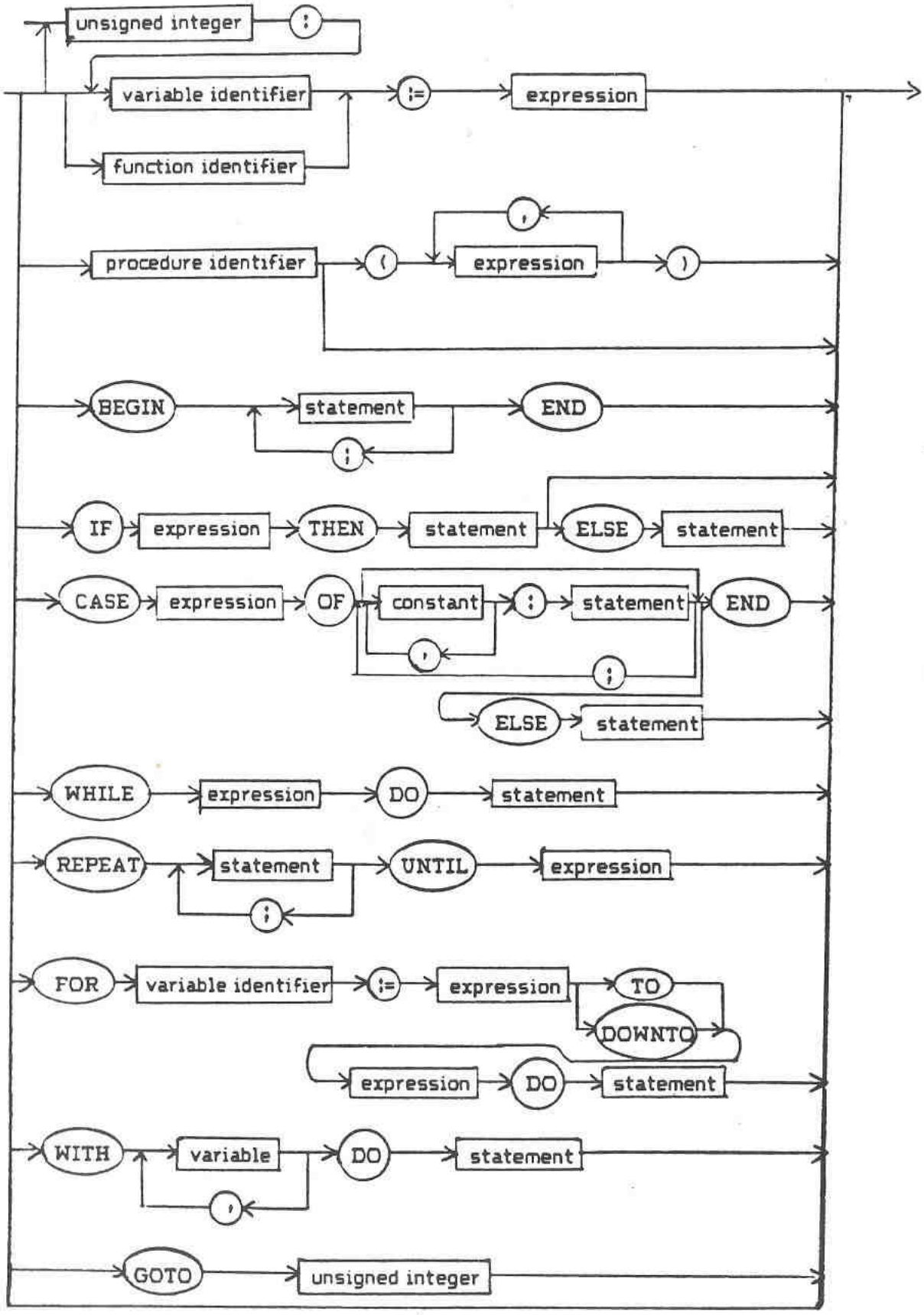
```
{Program to illustrate the use of TIN and TOUT.
 The program constructs a very simple telephone
 directory on tape and then reads it back. You
 should write any searching required.}

PROGRAM TAPE;

CONST
  Size=10;

TYPE
  Entry = RECORD
            Name : ARRAY [1..10] OF CHAR;
            Number : ARRAY [1..10] OF CHAR
          END;

VAR
  Directory : ARRAY [1..Size] OF Entry;
  I : INTEGER;

BEGIN

{Set up the directory..}

  FOR I:= 1 TO Size DO
  BEGIN
    WITH Directory[I] DO
    BEGIN
      WRITE('Name please');
      READLN;
      READ(Name);
      WRITELN;
      WRITE('Number please');
      READLN;
      READ(Number);
      WRITELN
    END
  END;

{To dump the directory to tape use..}

  TOUT('Director',ADDR(Directory),SIZE(Directory))

{Now to read the array back do the following..}

  TIN('Director',ADDR(Directory))

{And now you can process the directory as you wish.....}

END.
```

```
 10 {Program to list lines of a file in reverse order.
 20  Shows use of pointers, records, MARK and RELEASE.}
 30
 40 PROGRAM ReverseLine;
 50
 60 TYPE elem=RECORD                              {Create linked-list structure}
 70             next: ^elem;
 80             ch: CHAR
 90           END;
100      link=^elem;
110
120 VAR prev,cur,heap: link;                      {all pointers to 'elem'}
130
140 BEGIN
150  REPEAT                                       {do this many times}
160    MARK(heap);                                {assign top of heap to 'heap'.}
170    prev:=NIL;                                 {points to no varaible yet.}
180    WHILE NOT EOLN DO
190       BEGIN
200         NEW(cur);                            {create a new dynamic record}
210         READ(cur↑.ch);                       {and assign its field to one
220                                               character from file.}
230         cur↑.next:=prev;                     {this field's pointer adresses}
240         prev:=cur                            {previous record.}
250       END;
260
270 {Write out the line backwards by scanning the records
280  set up backwards.}
290
300    cur:=prev;
310    WHILE cur <> NIL DO                        {NIL is first}
320      BEGIN
330         WRITE(cur↑.ch);                      {WRITE this field i.e. character
340         cur:=cur↑.next                       {Address previous field.}
350      END;
360    WRITELN;
370    RELEASE(heap);                             {Release dynamic variable space.
380    READLN                                     {Wait for another line}
390  UNTIL FALSE                                  {Use CC to exit}
400 END.
```

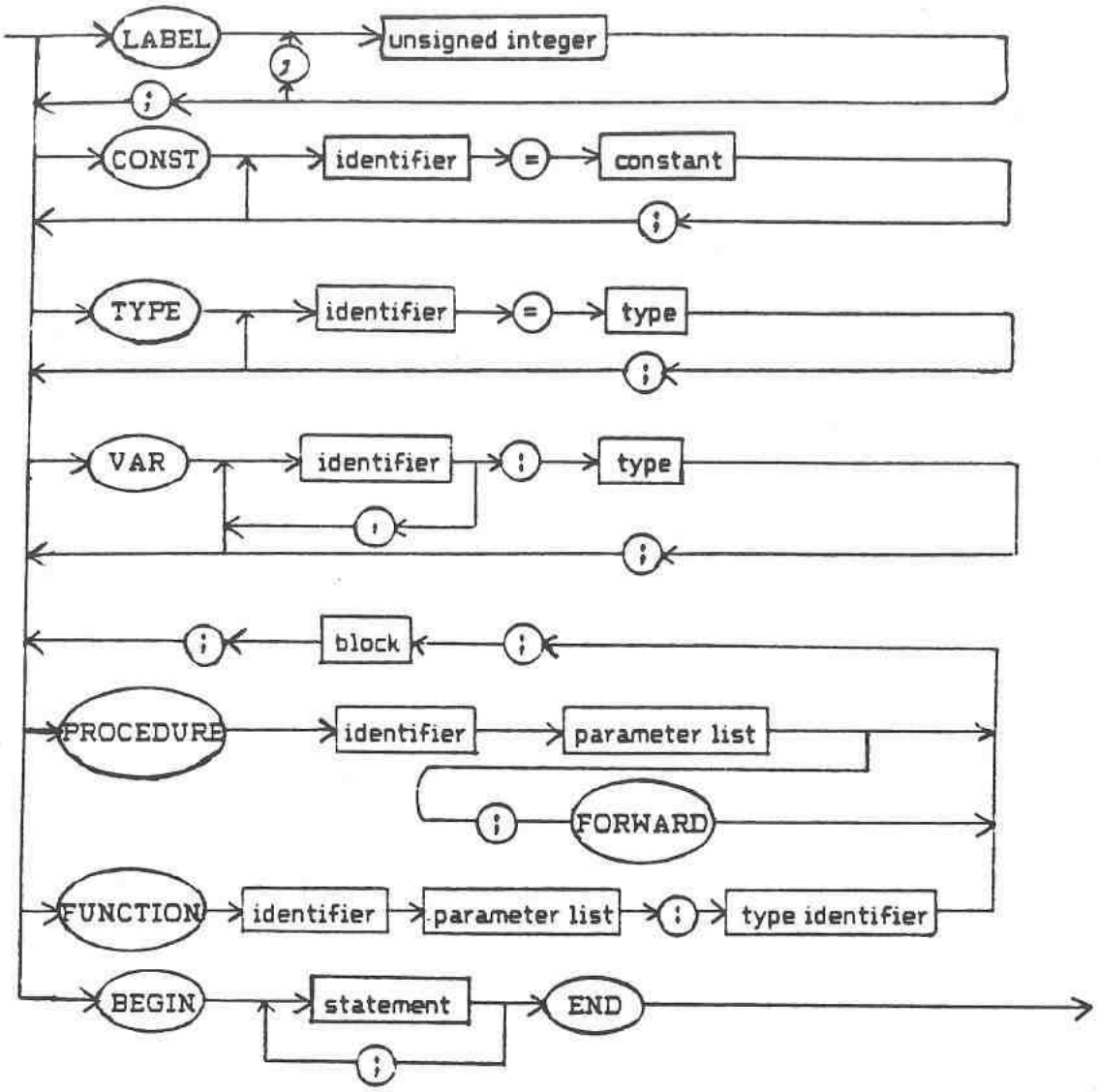
```
 10 {Program to show the use of recursion}
 20
 30 PROGRAM FACTOR;
 40
 50 {This program calculates the factorial of a number input from the
 60  keyboard  1) using recursion  and  2) using an iterative method.}
 70
 80 TYPE
 90   POSINT = 0..MAXINT;
100
110 VAR
120   METHOD : CHAR;
130   NUMBER : POSINT;
140
150 {Recursive algorithm.}
160
170 FUNCTION RFAC(N : POSINT) : INTEGER;
180
190  VAR F : POSINT;
200
210  BEGIN
220    IF N>1 THEN F:= N * RFAC(N-1)                    {RFAC invoked N times}
230           ELSE F:= 1;
240    RFAC := F
250  END;
260
270 {Iterative solution}
280
290 FUNCTION IFAC(N : POSINT) : INTEGER;
300
310   VAR I,F: POSINT;
320   BEGIN
330    F := 1;
340    FOR I := 2 TO N DO F := F*I;                     {Simple Loop}
350    IFAC:=F
360   END;
370
380 BEGIN
390   REPEAT
400     WRITE('Give method (I or R) and number    ');
410     READLN;
420     READ(METHOD,NUMBER);
430     IF METHOD = 'R'
440          THEN WRITELN(NUMBER,'! = ',RFAC(NUMBER))
450          ELSE WRITELN(NUMBER,'! = ',IFAC(NUMBER))
460   UNTIL NUMBER=0
470 END.
```

```
 10 {Program to show how to 'get your hands dirty'!
 20  i.e. how to modify Pascal variables using machine code.
 30  Demonstrates PEEK, POKE, ADDR and INLINE.}
 40
 50 PROGRAM divmult2;
 60
 70 VAR r:REAL;
 80
 90 FUNCTION divby2(x:REAL):REAL;              {Function to divide by 2 ..
100                                             .. quickly}
110 VAR i:INTEGER;
120 BEGIN
130  i:=ADDR(x)+1;                            {Point to the exponent of x}
140  POKE(i,PRED(PEEK(i,CHAR)));              {Decrement the exponent of x.
150                                             see Appendix 3.1.3.}
160  divby2:=x
170 END;
180
190 FUNCTION multby2(x:REAL):REAL;            {Function to multiply by 2..
200                                             .. quickly}
210 BEGIN
220  INLINE(£DD,£34,3);                      {INC (IX+3) - the exponent of
230                                             - see Appendix 3.2.}
240  multby2:=x
250 END;
260
270 BEGIN
280  REPEAT
290   WRITE('Enter the number r ');
300   READ(r);                                {No need for READLN - see
310                                            Section 2.3.1.4 }
320
330   WRITELN('r divided by two is',divby2(r):7:2);
340   WRITELN('r multiplied by two is',multby2(r):7:2)
350  UNTIL r=0
360 END.
```

STATEMENT.


unsigned integer
:
variable identifier
:=
expression
function identifier
,
procedure identifier
(
expression
)
BEGIN
statement
END
;
IF
expression
THEN
statement
ELSE
statement
CASE
expression
OF
constant
:
statement
END
,
;
ELSE
statement
WHILE
expression
DO
statement
REPEAT
statement
UNTIL
expression
;
FOR
variable identifier
:=
expression
TO
DOWNTO
expression
DO
statement
WITH
variable
DO
statement
,
GOTO
unsigned integer

1.16 BLOCK.


LABEL
unsigned integer
,
;
CONST
identifier
=
constant
;
TYPE
identifier
=
type
;
VAR
identifier
:
type
,
;
;
block
;
PROCEDURE
identifier
parameter list
;
FORWARD
FUNCTION
identifier
parameter list
:
type identifier
BEGIN
statement
END
;

Titlar i MicroSolverserien hösten -83

MicroSolver-serien är framtagen av Digilog. Vi har ansträngt oss att samla ihop ett brett sortiment s.k. nyttoprogram i en enhetlig serie, som betecknas av att alla manualer är bättre och utförligare än vad Du är van vid. Manualerna är naturligtvis på svenska. Programmen levereras i snygga pärmar där Du även kan spara Dina programlistningar och kommentarer. Själva programmen är inte vilka som helst, utan det allra bästa inom respektive område som finns på marknaden. Varje produkt är lika bra eller bättre än något annat som Du hittar på marknaden. Genom att vi tar fram stora serier av alla program kan vi också hålla priser som tål att jämföras!

Superfile

Ett mycket kraftfullt databas-program, som är helt anpassat till svenska förhållanden. All text och alla menyer i programmet är på svenska. De svenska tecknen ÅÄÖåäö finns tillgängliga på tangentbordet och printern. Du kan skapa upp till 36 olika sätt att visa data på, upp till 20 poster kan visas samtidigt på TV-skärmen och varje post kan innehålla upp till 4.600 tecken, men med normalstora poster kan man få in ungefär 250 olika poster samtidigt. Suveräna sökrutiner, möjlighet att skriva egna BASIC-rutiner som kan fogas till programmet, som i övrigt är skrivet helt i maskinkod och mycket snabbt. Endast 48k Spectrum

Pascal

Pascal är det mest använda programmeringsspråket inom teknik och naturvetenskap hos stordatorer. Den är uppbyggd på ett sådant sätt att överskådligheten blir maximal, samtidigt som programmet följer vanligt logiskt tänkande. Det är lättare att undvika fel, och skulle dessa ändå uppstå får man specifik hjälp i varje situation genom ett stort antal felmeddelanden. Digilog marknadsför den enda Pascal-kompilatorn till Spectrum.

Detta är den mycket välkända Hisoft Pascal i en variant för Sinclair Spectrum. I stort sett följer den Standard Jensen/Wirth Pascal, men har även vissa utökade möjligheter så som PEEK och POKE samt möjlighet att lägga in maskinkodsavsnitt i programmet. Dessutom finns rutiner för s.k. Turtle Graphics som standard på kassetten, d.v.s. möjlighet att med pixel-precision skapa Dina egna bilder (ungefär som med en ljuspenna, men Du flyttar markören med hjälp av tangentbordet). Dessutom har Pascalkompilatorn en radeditor som Du inte sett maken till på Spectrum. Search, replace, renumber och många andra nyttiga funktioner som merge är standard. All editering blir snabbare och effektivare! Endast 48k Spectrum.

SYS 64

Här har Du äntligen möjligheten att få fler tecken per rad till Din Sinclair Spectrum. Hela 64 tecken per rad, och alla tecken kan användas, även grafiska och tecken som Du själv skapat. Du kan köra Dina gamla BASIC-program med SYS 64. 32 och 64 tecken per rad kan blandas efter behag, varvid Du kan få t.ex. en stil på rubriker och en annan i texten. Ett måste för alla som jobbar med listor och tabeller och ett mycket bra hjälpmedel som ökar överskådligheten för Dig som sitter och skriver program. För 16 och 48k Spectrum.

BASIC-kompilator

Omvandlar Dina BASIC-program till maskinkod. Programmen blir då 10 till 100 gånger snabbare! Bra vid alla tillfällen, men ett måste för att få lite speed på annars långsamma spel som Du skrivit i BASIC. Tänk så skönt för Dig som har skrivit ett stort program som tar evigheter att köra, t.ex. sorteringsprogram och texthantering. Alla behöver en BASIC-kompilator. På marknaden finns ett flertal mer eller mindre lyckade varianter på temat. Varför chansa? Köp den BASIC-kompilator som utsågs till vinnare i "Your Computers" stora test (juninumret 1983).

Naturligtvis är det Digilog som säljer den! Som extra finess har den även "Mini Sprites Graphics", d.v.s. alla egna tecken som Du definierar (och även alla som redan finns, förstås) kan Du få att röra sig snabbt och ryckfritt (med pixelförflyttning) över skärmen med ett enkelt kommando, och Du kan styra dem hur Du vill.

16k FORTH

Först igen! Vi har en FORTH-kompilator som Du kan använda på en 16k Spectrum! Du har över 130 fördefinierade ord. Snabb och tillförlitlig. Inte svårare än BASIC, men kompileras så att programmet omvandlas till maskinkod. Mycket, mycket snabbare än BASIC! Bygg ut Din Spectrum till 48k, vänd på FORTH-kassetten och där finner Du 48k flyttals-FORTH! Du får alltså dubbel valuta för pengarna! Läs om flyttals-FORTH här nedan.

48k flyttals-FORTH

Visst finns det andra FORTH för 48k Spectrum. Det finns dock en liten hake med dessa: de kan bara behandla heltal! Det är ju inte så användbart i alla sammanhang. Digilog presenterar den första FORTH:en till Spectrum som tillåter Dig att använda alla tal som Du kan använda i Spectrum BASIC. Inte nog med det! Detta är den enda FORTH till Spectrum som till fullo uppfyller FORTH-79 standard, de krav som ställs på FORTH för stordatorer. över 190 fördefinierade ord och en utmärkt, lättskött editor. På baksidan av kassetten finns dessutom marknadens enda FORTH-kompilator för 16k Spectrum. Har Du ännu inte byggt ut Din Spectrum till 48k, kan Du ändå köpa denna kassett. När Du sedan en dag kompletterar datorns minne får Du tillgång till flyttals-FORTH, men innan dess kan Du redan öva Dig att skriva program i det nya, populära språket!

DevPac

Assembler, disassembler, editor och maskinkodsmonitor i ett paket! Detta till samma pris som Du i annat fall skulle fått betala för enbart assemblern.

Nu är detta ingen "lågprisprodukt", utan tvärtom den allra bästa assembler-disassembler som finns på marknaden (det är inte vi som tycker det, utan en enad kritikerkår i engelsk datorpress). DevPac kommer från Hisoft i England. Alla som intresserat sig för assemblerprogrammering har redan hört namnet. Vi distribuerar den i Sverige, med en mycket tjock och utförlig manual.

Editorn är värd ett extra omnämnande. Du har nämligen automatisk SEARCH och REPLACE, vilket är mycket viktigt i assembler, eftersom man ofta vill ändra en adress som man använt på många ställen i ett program. Ett av de vanligaste och mest svårupptäckta felen är att man glömt byta adressen på ett eller ett par ställen, varvid hela programmet spårar ur. Detta undviker Du med denna assembler, eftersom editorn själv letar reda på den adress eller "label" som ska bytas. Inte bara på ett ställe i programmet utan på alla, och byter ut den mot den nya adressen. Ett enkelt och sinnrikt system hjälper Dig dessutom att söka efter syntaxfel. Blockförflyttningar är mycket enkelt. Ett obegränsat antal BREAKPOINTS kan sättas. Jämför med andra assemblers, som ofta tillåter bara en. Dumpa register, variabler och annat.