# JRT-Pascal

## A. Urheberrechtliche Fragen

Der Pascal-Compiler JRT-Pascal wurde von James Robert Tyson entwickelt. Der Texteditor ED wurde in seiner Urversion von Kazuo und Kazuko Nakazato in Japan erstellt. Das Programm war in dieser Version ausschließlich auf dem Computer NEC-8001 lauffähig, weil es direkt in den Bildschirmspeicher schreibt.

Die Bildschirmmeldungen der Programme wurden von Martin Kotulla ins Deutsche übersetzt. Außerdem wurden einige Programmänderungen durchgeführt, die die Bedienung vereinfachen beziehungsweise wesentliche Programmteile auf dem Schneider-CPC und Joyce überhaupt erst lauffähig machen. Die Programme bleiben weiterhin in der amerikanischen Originalversion in der Public-Domain und dürfen in der Originalversion ohne Einschränkungen kopiert und weiterverbreitet werden.

Für die deutsche Version der Programme gilt, daß ihre kommerzielle Verbreitung ohne schriftliche Einwilligung von Martin Kotulla unzulässig ist. Es wird ein Urheberrechtsanspruch auf die Programmanpassungen geltend gemacht. Der Preis des Gesamtpakets ist so niedrig angesetzt, daß sich wirklich jeder die Programme leisten kann.

Die Beschreibung, die Sie gerade lesen, ist voll urheberrechtlich geschützt und darf keinesfalls weiterverbreitet werden - weder unverändert noch in Abwandlungen. Die Verfolgung von Verstößen in zivil- und strafrechtlicher Hinsicht bleibt uns vorbehalten.

## B. Haftungsausschluß

Es wird keinerlei Haftung oder Gewährleistung dafür übernommen, daß die Programme oder das Handbuch fehlerfrei oder für bestimmte Zwecke geeignet sind. Ferner besteht für uns keine Pflicht, irgend jemanden über Änderungen an den Programmen oder der Dokumentation zu benachrichtigen.

## C. Der Texteditor ED

### 1. Grundlegendes über ED:

ED wurde in Turbo-Pascal geschrieben. Der Editor arbeit bildschirmorientiert und bietet eine sinnvolle Teilmenge der Befehle des Textprogramms WordStar. ED kann Texte bis zur Größe des RAM-Speichers abzüglich der Länge des Programmcodes bearbeiten. Das sind unter CP/M Plus oder mit Speichererweiterung unter CP/M 2.2 etwa 31000 Zeichen.

Es werden zwei Versionen des Editors geliefert:

- EDLARGE.COM für CP/M 2.2 mit Vortex-Speichererweiterung
- EDPLUS.COM für CP/M Plus auf dem CPC-6128 und Joyce

Es ist empfehlenswert, die für Sie geeignete Version in ED.COM umbenennen.

Fehlt beim Start von ED der Name der zu bearbeitenden Datei, fragt ED automatisch an:

```
A>ED
Dateiname? TEXT.TXT
```

Eine praktische Kurzform davon sieht so aus:

```
A>ED TEXT.TXT
```

ED kann sowohl neue als auch bereits bestehende Dateien bearbeiten. Der Editor erkennt den Unterschied automatisch und zeigt im entsprechenden Fall die Meldung *Neue Datei* an.

Der Editor kann in Dateien eventuell enthaltene TAB-Zeichen (ASCII-Code 9) nicht verarbeiten. Übernehmen Sie also Dateien aus anderen Textsystemen, sollten Sie vorher mit PIP die TABs durch Leerzeichen ersetzen:

```
A>PIP OHNE.TAB=MIT.TAB[T8]
```

## 2. Die Tastenbefehle von ED

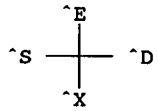
ED arbeitet mit einer Untermenge der WordStar-Befehle. Die Cursortasten und die <COPY>-Taste des Schneider-CPC werden sinnvoll genutzt. <COPY> dient zum Umschalten zwischen dem Überschreibe- und Einfügemodus (`Control-V`). Sie sollten möglichst stets im Überschreibemodus arbeiten, weil Sie dort Texte sehr viel schneller eingeben können und sich der Cursor auch schneller fortbewegen läßt.

ED startet im Überschreibemodus. Daß die erste Zeile etwas "zäh" einzugeben ist, ist ganz normal, weil ED erst den Speicher organisieren muß. Ab der zweiten Zeile geht das aber sehr schnell.

Ähnlich dem Editor von Turbo-Pascal arbeiten die Befehle wie `Cursor links`, `Cursor rechts`, <DEL> und `Control-G` nicht über Zeilengrenzen hinweg. Sie können aber dennoch Zeilen auftrennen und zusammenfügen. Dazu verwenden Sie `Control-N` (Auftrennen einer Zeile an der Cursorposition) und `Control-T` (Löschen des nächsten Wortes beziehungsweise des Zeilenumbruchs).

## 3. Cursorsteuerung

Der Cursor kann frei auf dem Bildschirm bewegt werden. Dazu dienen die normalen Cursortasten sowie das von WordStar bekannte Cursor-Kreuz:




Der Wort-Tabulator nach links wird mit `Control-A`, der nach rechts mit `Control-F` aufgerufen. Alternativ können `Shift-Linkspfeil` und `Shift-Rechtspfeil` benutzt werden.

Der Cursor kann auch über die Bildschirmgrenzen hinwegbewegt werden. Der Editor scrollt dann automatisch den Bildschirm in die entsprechende Richtung. Anstatt den Cursor zu bewegen, kann aber auch der Bildschirm unter dem Cursor "hinweggerollt" werden. Dazu dienen die Tasten `Control-W` und `Control-Z`.

Ein seitenweises Blättern durch den Text bieten die Tasten `Control-C` (Seite nach unten) und `Control-R` (Seite nach oben). Diese Funktionen können auch durch `Shift-Hochpfeil` und `Shift-Abwärtspfeil` aufgerufen werden.



Eine Zeile wird einfach durch Beschreiben mit Text eingegeben. Um in die nächste Zeile zu gelangen, können Sie am Zeilenende <ENTER> oder <RETURN> drücken. Dabei wird der Cursor nicht auf die erste Zeichenstelle gesetzt, sondern auf den Anfangsbuchstaben des ersten Wortes. Ist die Zeile leer, wird der Cursor unter das erste Zeichen der darüberstehenden Zeile gesetzt. Dies ist eine besonders praktische Eigenschaft des Editors, die die Eingabe von Programmen in strukturierten Sprachen wie Pascal oder C stark vereinfacht. Denn Einrückungen im Programmtext werden damit automatisch durchgeführt.

Erweiterte Cursorbefehle werden mit zwei Tastendrücken aufgerufen. Der erste ist stets `Control-Q`, darauf folgt eine weitere Taste. So bewegt `Control-Q Control-R`, im folgenden kurz als `^Q^R` bezeichnet, den Cursor an den Textanfang, `^Q^C` an das Textende. `^Q^S` setzt die Schreibmarke an den Anfang der Bildschirmzeile, `^Q^D` an ihr Ende.

Diese Kommandos lassen sich aber einfacher durch die Befehlskombination `Control` plus Cursortaste ersetzen:

| | | |
|---|---|---|
| `^Q^S` | `Control-Linkspfeil` | Cursor an den Zeilenanfang |
| `^Q^D` | `Control-Rechtspfeil` | Cursor an das Zeilenende |
| `^Q^R` | `Control-Hochpfeil` | Cursor an den Textanfang |
| `^Q^C` | `Control-Abwärtspfeil` | Cursor an das Textende |

## 4. Einfügen und Löschen

ED kann sowohl im Einfüge- als auch im Überschreibemodus arbeiten. Im erstgenannten werden neue Zeichen in eine Zeile eingesetzt, wobei die rechts neben ihnen stehenden um eine Position nach rechts geschoben werden. Im Überschreibemodus geht das vorher an der Cursorposition stehende Zeichen verloren. Die Umschaltung zwischen den beiden Modi geschieht über <COPY> oder `Control-V`.

<DEL> kann zum Löschen des Zeichens links vom Cursor benutzt werden, während `Control-G` das Zeichen unter dem Cursor löscht (vergleichen Sie <CLR> im Schneider-Basic).

Weitere Befehle:

| | |
|---|---|
| `^T` | das rechts folgende Wort löschen (arbeitet auch über Zeilengrenzen). |
| `^B` | eine Leerzeile über der Zeile des Cursors einfügen. |
| `^N` | eine neue Zeile beziehungsweise einen Zeilenbruch einfügen. |
| `^Y` | die Zeile, in der der Cursor steht, löschen. |
| `^Q^Y` | von der Cursorposition bis zum Zeilenende löschen. |

## 5. Blockbefehle

Wollen Sie eine größere Anzahl von Zeilen löschen, kopieren, bewegen oder in eine Datei schreiben, müssen Sie einen Block **markieren**. Dazu fahren Sie mit dem Cursor den Blockanfang an und drücken `^K^B`. Dann gehen Sie an das Blockende und tippen `^K^K`. Der durch die beiden Zeichen begrenzte Block wird nun invers dargestellt. Sie können einen der folgenden Blockbefehle anwenden:

| | |
|---|---|
| `^K^C` | Block an die neue Cursorposition kopieren |
| `^K^V` | Block an die aktuelle Cursorposition bewegen |
| `^K^Y` | Block löschen |
| `^K^W` | Block als Diskettendatei schreiben |

Desweiteren macht `^K^H` einen markierten Block wieder unsichtbar. `^K^R` liest einen Textblock von der Diskette ein, der aber nicht markiert wird. Bitte setzen Sie stets den Blockanfang vor dem Blockende und setzen Sie nicht gleichzeitig mehrere Blockmarkierungen!

### 6. Weitere Editierbefehle

Wenn Sie eine Datei bearbeitet haben, können Sie ^K^D drücken. ED zeigt ein Menü in der obersten Bildschirmzeile an:

<S> Speichern der Datei (vergleiche ^K^D bei WordStar)
<E> Ende ohne Speichern (vergleiche ^K^Q bei WordStar)
<N> Neues Bearbeiten der Datei (wird erneut geladen)
<Z> Zurück zum Text

^Q^F: Zeichenkette (bis 40 Zeichen) suchen.

^Q^A: Suchen und ersetzen. An der ersten Fundstelle erscheint *Ersetzen? (J/N)*. Nach der nächsten Fundstelle suchen Sie mit Control-L.

^L: Letztes Suchen oder Ersetzen wiederholen.

^Q^Z: Ein- und Ausschalten der aktuellen Cursorpositionen in der obersten Zeile. Bei ausgeschalteter Anzeige geht das Editieren schneller.

## D. Historisches zu JRT-Pascal:

James Robert Tyson (J-R-T) bot den Pascal-Compiler JRT-Pascal im Jahr 1982 durch Anzeigen in amerikanischen Computerzeitschriften zum damals sagenhaften Preis von 29,95 Dollar an. Was sich dann ereignete, hielt kaum jemand für möglich. Tysons Firma JRT-Systems erhielt über einhunderttausend Bestellungen! Als Einmannbetrieb war JRT-Systems natürlich völlig überfordert, alle Bestellungen zu bearbeiten. Tyson bekam deshalb Ärger mit der amerikanischen Bundeshandelskommission, weil er bezahlte Waren nicht liefern konnte. Tyson mußte seine Firma schließen und gab den Compiler in die Public-Domain.

## E. Die Funktionsweise von JRT-Pascal:

JRT-Pascal arbeitet im klassischen Compiler-Zyklus: Editieren, Compilieren, Starten, Editieren ... Der Compiler JPASCAL.COM erzeugt keinen echten Maschinencode, sondern einen sehr maschinennahen Zwischencode. Das hat den Vorteil, daß die Programmodule keine absoluten Speicheradressen benötigen, sondern frei verschiebbar sind. Außerdem belegen sie nur einen Bruchteil des Platzes echter Maschinenprogramme. So können Sie viel längere Programme entwickeln. Das Laufzeitmodul EXEC.COM lädt die benötige Zwischendatei von der Diskette und startet das compilierte Programm.

Im Zwischenformat, das an der Erweiterung .INT im Dateinamen erkennbar ist, werden auch verschiedene Hilfsprogramme geliefert, die die Erstellung von Pascal-Programmen erheblich vereinfachen. Zu ihnen kommen wir später noch.

Die Programme sind in ihrer Länge nicht auf den RAM-Hauptspeicher beschränkt, sondern können bis zur Kapazität aller angeschlossenen Diskettenstationen erweitert werden. Bei Bedarf lädt das Laufzeitmodul EXEC.COM automatisch die richtigen Prozeduren in den Speicher und führt sie aus. Im Gegensatz zum recht simplen Overlay-System anderer Compilersprachen verwendet JRT-Pascal ein ausgeklügeltes Verfahren zur Verwaltung der Prozeduren, das **External-System**. Es lädt die benötigten Prozeduren nur einmal in den Speicher. Werden sie später wieder gebraucht, müssen sie nicht erneut geladen werden. Das spart enorm Abarbeitungszeit. Erst wenn der Platz im RAM knapp wird, stellt JRT-Pascal fest, welche Prozedur schon am längsten nicht mehr benötigt wurde, und löscht sie. Dadurch findet eine neue Prozedur Platz im Speicher.



## F. Anpassung von JRT-Pascal:

JRT-Pascal kommt mit einem einzelnen Diskettenlaufwerk aus. Dennoch ist es natürlich angenehmer, mit mehreren Floppystationen zu arbeiten. Man kann deshalb JRT-Pascal mitteilen, welche Laufwerke angeschlossen sind und in welcher Reihenfolge sie nach bestimmten Dateien abgesucht werden sollen. Bevor Sie den Compiler das erste Mal starten können, müssen Sie diese kleine Aufgabe erledigen.

Das ist ganz einfach. Stellen Sie sich vor, Sie haben eine Floppystation als A: und die RAM-Diskette der Speichererweiterung als C:. Bevorzugt soll JRT-Pascal natürlich auf der RAM-Disk suchen, weil diese viel schneller arbeitet als eine normale Floppy. Starten Sie also die Anpassungsroutine CUSTOMIZ.INT:

```
A>EXEC CUSTOMIZ

** EXEC V2D **
Customize V2.0D
Neue Disketten-Suchliste: CA
Programmende ...
```

Beim Joyce würde die Suchliste MA oder MAB lauten, da die RAM-Disk hier als M: angesprochen wird.

## G. Der Sprachumfang von JRT-Pascal

JRT-Pascal lehnt sich eng an den von Niklaus Wirth, dem "Erfinder" von Pascal, festgelegten Sprachumfang an. Es gibt nur kleine Einschränkungen, sehr wohl aber komfortable Erweiterungen des Standard-Pascal.

### 1. Bezeichner (Identifier)

Unter Bezeichnern versteht man die Namen von Variablen, Prozeduren, Datenfelder etc. Sie dürfen bis zu 64 Zeichen umfassen, von denen alle signifikant sind. Die Namen können aus Buchstaben, Ziffern, dem Dollarsymbol und dem Unterstrich bestehen. Das erste Zeichen muß aber ein Buchstabe sein. Außerdem dürfen Bezeichner nicht mit den Standardbezeichnern wie FOR, BEGIN oder TO übereinstimmen.

### 2. Erweiterter CASE-Befehl

Das Standard-Pascal sieht bei CASE kein definiertes Verhalten vor, wenn keiner der CASE-Werte zutrifft. Bei JRT-Pascal können Sie eine ELSE-Klausel definieren, die in diesem Fall aufgerufen wird:

```
CASE a OF
   2:    WriteLn('2');
   4:    WriteLn('4');
   ELSE: WriteLn('Weder 2 noch 4');
END;
```

### 3. Kommentare

Kommentare werden in geschweifte Klammern eingeschlossen. Alternativ kann auch das zweite übliche Format verwendet werden:

```
{ Das ist ein Kommentar in JRT-Pascal }
(* So geht es aber auch *)
```

## 4. Variablentypen und Wertebereiche

- **Integerzahlen** besitzen einen Wertebereich von -32768 bis 32767. Sie dürfen dezimal oder hexadezimal eingegeben werden. Hexwerte erkennt JRT-Pascal an einem an die Zahl angehängten <H>. Beginnt ein Hexwert mit einem Buchstaben (A, B, C, D, E oder F), muß ihm eine Null vorangestellt werden.

Beispiele: `32000, 7FFFH, 0F000H, -0A0H`
Definition: `VAR a:INTEGER;`

- **Realzahlen** sind Fließkommazahlen mit 14stelliger Genauigkeit. Ihr Exponent bewegt sich im Bereich von -64 bis 63.

Beispiele: `3.1415926, -32.44E-30, 1.0`
Definition: `VAR x:REAL;`

- **Zeichenvariablen** bestehen aus einem Buchstaben, dessen ASCII-Wert zwischen 0 und 255 liegt.

Beispiele: `'a', ' ', '*'`
Definition: `VAR c:CHAR;`

Es wird keinerlei PACKing unterstützt, weil es auf 8-Bit-Computern nicht sinnvoll ist.

- **Boolesche Variablen** können die logischen Werte TRUE und FALSE annehmen. Sie werden zum Speichern von Wahrheitswerten benutzt.

Definition: `VAR b:BOOLEAN;`

- **Zeichenketten** sind als dynamisch verwaltete Strings verfügbar. Strings dürfen bis zu 65535 Zeichen umfassen, was natürlich durch den freien Speicher in CP/M beschränkt wird. Strings sollten mit einer Längenangabe deklariert werden. Fehlt diese, wird eine Stringlänge von 80 Zeichen entsprechend einer Bildschirmzeile angenommen.

```
Beispiel:    'Ein String'
Definition:  VAR s:STRING;          (* 80 Zeichen *)
             VAR t:STRING[500];     (* 500 Zeichen *)
```

Strings dürfen bei allen Gelegenheiten in JRT-Pascal verwendet werden - mit einer Ausnahme: Diskettenzugriffe sind mit Strings nicht möglich. Hier muß die Umschreibung ARRAY OF CHAR benutzt werden. Durch Funktionen und Prozeduren wie CONCAT, COPY, DELETE, LENGTH, INSERT und POS wird ein sehr komfortables Arbeiten mit Strings möglich.

- **Datenfelder** bestehen aus einer Vielzahl von Variablen eines Grundtyps. So ist ein ARRAY[1..100] OF INTEGER in der Lage, 100 Integerzahlen zu speichern. Bis zu acht Dimensionen werden bei Arrays unterstützt.

```
Beispiel:    a[23]:=256+a[22];
Definition:  VAR x:ARRAY [40..555] OF CHAR;
```

Die CONCAT-Funktion darf auch bei ARRAYs benutzt werden.



- **Mengen** besitzen stets eine Länge von 16 Bytes und sind bitweise gepackt. Sie können auf Diskette nur in Binärdateien gespeichert werden.

```
Beispiel:    TYPE ascii = SET OF 'A'..'Z';
             VAR buchst:ascii;
             buchst:=['A'..'Z'];
             IF 'A' IN buchst THEN WriteLn('A ist vorhanden!');
```

- **Zeiger** enthalten die virtuelle Adresse dynamischer Variablen. Dynamische Variablen werden mit NEW erzeugt oder sind scheinbare Variablen von MAP, die einen Zugriff auf den Speicher gestatten (siehe Abschnitt M).

Bei den Adressen, die die Pointer enthalten, handelt es sich nicht um die tatsächlichen Speicheradressen. Letztere können Sie über die Funktion ADDR ermitteln: `wirkliche_adresse:=ADDR(zeiger^);`

**Records** erfüllen vom Prinzip her ähnlich Aufgaben wie Datenfelder. Sie enthalten mehrere Variablen, die aber unterschiedliche Datentypen besitzen können.

```
Beispiel:    politiker.beruf:='Bundeskanzler';
             politiker.gehalt:=12000;

Definition:  VAR x:RECORD
                    a,b:REAL;
                    c:CHAR;
                  END;
             (* Variante Records: *)
             VAR y:RECORD
                    CASE INTEGER OF
                     0:(zahl:REAL);
                     1:(zeichen:ARRAY [1..8] OF CHAR);
                  END;
```

## H. Das Kompilieren und Starten von Programmen

Ein typischer Programmierzyklus sieht bei JRT-Pascal so aus: Zuerst erstellen Sie mit dem Editor ED.COM ein Pascal-Programm mit der Namenserweiterung .PAS:

```
A>ED DEMO.PAS
```

Nehmen wir an, Sie hätten das folgende kleine Programm eingegeben:

```
PROGRAM Demo;

VAR i:INTEGER;

BEGIN
  FOR i:=1 TO 10 DO Write(i,' ');
END.
```

Sobald Sie das Editieren beendet haben, können Sie JPASCAL.COM aufrufen. Dieser erzeugt aus DEMO.PAS die Zwischendatei DEMO.INT:

```
A>JPASCAL DEMO

JRT Pascal  V2.0D
```

```
Copyright 1982 JRT Systems

0000  0001:         PROGRAM Demo;
0000  0002:
0003  0003:         VAR i:INTEGER;
0003  0004:
0006  0005:         BEGIN
0025  0006:           FOR i:=1 TO 10 DO Write(i,' ');
0026  0007:         END.
0026  0008:
 0 Fehler entdeckt
Modulgroesse: 42 Dez.Bytes
Kompilation fertig DEMO
```

Da der Compiler keine Fehler gemeldet hat, können Sie das Programm sofort starten:

```
A>EXEC DEMO

** EXEC V2D **

1 2 3 4 5 6 7 8 9 10

Programmende ...
A>
```

## I. Eingebaute Prozeduren und Funktionen

Die folgenden Prozeduren und Funktionen sind direkt in JRT-Pascal integriert. Sie können ohne jede vorherige Deklaration sofort benutzt werden.

### ABS - Absoluter Betrag einer Zahl _____ Funktion

Syntax: `ABS(integerausdruck)`
`ABS(realausdruck)`

Die ABS-Funktion ermittelt den absoluten Betrag einer Zahl.

Beispiele:
```
a:=ABS(-3);
WriteLn('Der absolute Betrag von -40 ist ',ABS(-40));
```

### ADDR - Speicheradresse einer Variablen _____ Funktion

Syntax: `ADDR(variable)`

Die ADDR-Funktion ermittelt die wirkliche Speicheradresse einer Variablen, eines Feldelements, eines Recordfelds oder einer dynamischen Variablen. Beachten Sie, daß sich die Speicheradresse dynamischer Variablen während der Programmabarbeitung ändern kann.

Beispiele:
```
WriteLn('Die Adresse der Variablen x ist ',ADDR(x));
ad:=ADDR(basis^);
```

### CHR - ASCII-Code einer Integerzahl _____ Funktion

Syntax: `CHR(integerausdruck)`



Die CHR-Funktion liefert das zu einer Integerzahl gehörende ASCII-Zeichen. Sie kann dazu benutzt werden, Steuerzeichen an den Drucker oder den Bildschirm zu senden.

Beispiele:
```
IF cpmplus THEN Write(CHR(27),'E') ELSE Write(Chr(12));
x:=CHR(a);
```

### CONCAT - Verkettung von Strings _____ Funktion

Syntax: `CONCAT(strausdruck_1,strausdruck_2,...,strausdruck_n)`

Die CONCAT-Funktion verbindet mehrere dynamische Strings, Stringkonstanten oder strukturierte Variablen miteinander. Sie gibt einen dynamischen String mit der benötigten Länge zurück.

Beispiele:
```
Gesamt:=CONCAT(Vorname,' ',Name,' ',Strasse,' ',Ort);
Write(CONCAT('JRT','-','Pascal');
```

Als Kurzform darf aber auch das Pluszeichen benutzt werden:

```
Gesamt:=Vorname+' '+Name+' '+Strasse+' '+Ort;
Write('JRT'+'-'+'Pascal');
```

### COPY - Kopieren von Stringsegmenten _____ Funktion

Syntax: `COPY(strausdruck,startposition,länge)`

Die COPY-Funktion gibt einen Stringausdruck zurück, der einen Teilstring von <strausdruck> enthält. Es werden <länge> Zeichen ab <startposition> extrahiert.

Beispiele:
```
Write(COPY('JRT-Pascal',5,6));
a:=COPY('ABC',2,1));
```

### DELETE - Zeichen aus String herauslöschen _____ Prozedur

Syntax: `DELETE(stringvariable,position,länge)`

Die DELETE-Prozedur entfernt aus dem Inhalt einer Stringvariablen eine bestimmte Anzahl von Zeichen. Beginnend mit <position> werden <länge> Zeichen gelöscht.

Beispiele:
```
DELETE(st,25,6);
DELETE(s,a,b);
```

### DISPOSE - Speicher für dynamische Variablen freigeben __ Prozedur

Syntax: `DISPOSE(zeigervariable);`

Den zuvor mit NEW für eine dynamische Variable bereitgestellten Speicherplatz kann man mit DISPOSE wieder frei verfügbar machen. JRT-Pascal führt dann bei Bedarf automatisch eine Neuorganisation des Speichers durch.

Beispiele:
```
pointer:=^dynvariable; NEW(pointer); DISPOSE(pointer);
```

```
DISPOSE(a);
```

## FILLCHAR - Strukturvariable mit Bytes füllen _____Prozedur

Syntax: `FILLCHAR(strukvariable,länge,zeichen)`

Die FILLCHAR-Prozedur kann dazu benutzt werden, Datenfelder und Records sehr schnell auf einen bestimmten Wert zu initialisieren. <länge> gibt die Zahl der zu ändernden Bytes an, <zeichen> das Byte, mit dem gefüllt wird. Bei Verwendung der Prozedur FILLCHAR kann JRT-Pascal keine Plausibilitätsprüfung durchführen. Achten Sie also unbedingt darauf, keine systemrelevanten Speicherbereiche zu überschreiben!

Beispiele:
```
FILLCHAR(feld1,500,CHR(0));
FILLCHAR(record5,1000,'x'));
```

## FREE - Freien Speicherplatz bestimmen _____Funktion

Syntax: `FREE`

Die FREE-Funktion ermittelt den zu einem bestimmten Zeitpunkt verfügbaren Speicherplatz. Sind mehr als 32K frei, wird ein negativer Wert zurückgegeben, da Integerzahlen in JRT-Pascal mit Vorzeichen dargestellt werden. Der wahre Wert ist dann `65536.0+FREE`. Da die Laufzeitbibliothek inaktive externe Prozeduren löschen kann, ist oft viel mehr Speicher vorhanden, als gemeldet wird.

Beispiele:
```
WriteLn('Es sind ',FREE,' Bytes verfügbar');
a_real:=FREE; IF a_real<0 THEN a_real:=65536.0+a_real;
```

## HEX$ - Umwandlung einer Zahl in einen Hexstring _____Funktion

Syntax: `HEX$(variable)`

Die HEX$-Funktion wandelt eine Variable beliebigen Typs in einen hexadezimalen String mit der doppelten Länge um.

Beispiele:
```
Write(HEX$(3.1415926));
x:=HEX$(16383);
```

Entsprechend den Z80-Konventionen wird das Lowbyte vor das Highbyte gestellt. Ist das - besonders bei 16-Bit-Werten - nicht gewünscht, können Sie statt HEX$ die folgende Umwandlungsprozedur benutzen:

```
FUNCTION HexConvert(x:INTEGER):STRING[4];
VAR a:STRING[4];
BEGIN
  a:=HEX$(x);
  HexConvert:='    ';
  HexConvert[1]:=a[3];
  HexConvert[2]:=a[4];
  HexConvert[3]:=a[1];
  HexConvert[4]:=a[2];
END;
```

## INSERT - Zeichen in String einfügen _____Prozedur



Syntax: `INSERT(einfügestring,zielstring,startposition)`

Die Prozedur INSERT setzt den <einfügestring> in den <zielstring> ein und beginnt damit bei <startposition>. Der Zielstring muß stets eine Variable sein.

Beispiele:
```
s:='JRTPascal'; INSERT('-',s,4);
INSERT(a,b,3);
```

## LENGTH - Stringlänge ermitteln _____Funktion

Syntax: `LENGTH(dynam_stringvariable);`

Die LENGTH-Funktion ermittelt die Länge eines dynamischen Strings. LENGTH darf ausschließlich bei dynamischen Strings benutzt werden.

Beispiele:
```
s:='ABCD'; Write(LENGTH(s));
a:=''; Write(LENGTH(a));
```

## NEW - Einer dynamischen Variablen Speicherplatz zuweisen _Prozedur

Syntax:
```
NEW(zeigervariable);
NEW(zeigervariable,feld_1,feld_2,...,feld_n);
```

Die NEW-Prozedur stellt Speicherplatz für dynamische Variablen zur Verfügung. Sie übergibt dessen virtuelle Adresse an die Zeigervariable. Die Feld-Angaben sind für variante Records gedacht und korrespondieren mit den Feldern in CASE bei der Deklaration des Records.

Beispiele:
```
NEW(pointer);
NEW(pointer,f1,f2);
```

## ODD - Feststellen, ob eine Zahl ungerade ist _____Funktion

Syntax: `ODD(integerausdruck);`

Die ODD-Funktion gibt den Wahrheitswert TRUE zurück, wenn der Integerausdruck gerade ist. Sonst gibt sie FALSE zurück.

Beispiele:
```
Write(ODD(1));
Write(ODD(2));
```

## ORD - ASCII-Code eines Zeichens bestimmen _____Funktion

Syntax: `ORD(zeichenausdruck);`

Die ORD-Funktion ermittelt den zu einem ASCII-Zeichen gehörenden Zahlencode. Bei Strings wird deren erstes Zeichen zur Berechnung herangezogen.

Beispiele:
```
Write(ORD('A'));
x:=ORD('JRT');
```

## PORTIN - Z80-Ports lesen _____ Funktion

Syntax: `PORTIN(integerausdruck);`

Die PORTIN-Funktion liest ein Byte von einem der Z80-Ports. Der zurückgegebene Wert ist vom Typ CHAR. PORTIN kann beim Schneider-CPC nicht sinnvoll benutzt werden, weil es auf 8-Bit-Portadressen ausgerichtet ist. Beim Joyce hingegen bereitet PORTIN keine Probleme.

Beispiele:
```
x:=PORTIN(0FH);
WHILE PORTIN(console)=CHR(0) DO;
```

## PORTOUT - Auf Z80-Ports schreiben _____ Prozedur

Syntax: `PORTOUT(portnummer,wert);`

Die Prozedur PORTOUT schreibt auf einen der Z80-Ports den angegeben Wert, der vom Typ CHAR oder STRING sein muß. Ähnlich dem bei PORTIN Gesagten, ist auch diese Prozedur beim Schneider-CPC nur mit Einschränkungen verwendbar.

Beispiele:
```
PORTOUT(0FFH,CHR(80H));
PORTOUT(diskmotor,CHR(0));
```

## POS - Zeichenmuster in Strings suchen _____ Funktion

Syntax:
```
POS(suchmuster,string);
POS(suchmuster,string,startposition);
```

Durchsucht den String nach dem ersten Vorkommen des <suchmusters> und übergibt die Fundstelle. Wahlweise kann eine Startposition spezifiziert werden.

Beispiele:
```
x:=POS('AN','BANANE');
y:=POS('AN','BANANE',4);
```

## PRED - Vorgänger ermitteln _____ Funktion

Syntax:
```
PRED(integerausdruck);
PRED(zeichenausdruck);
```

Die PRED-Funktion ermittelt den Vorgänger einer Integerzahl oder eines ASCII-Zeichens. Der Vorgänger von 10000 ist 9999, von 'b' ist er 'a'.

Beispiele:
```
Write(PRED('b'));
Write(PRED(10000));
```

## REAL$ - Realzahl in formatierten String wandeln _____ Funktion

Syntax: `REAL$(realausdruck);`

Die REAL$-Funktion wandelt eine REAL-Zahl in einen druckbaren String mit 22 Zeichen Länge um. Das Format dafür ist " +0.12345678901234E+00".

Beispiele:
```
a:=REAL$(3.141592653);
b:=REAL$(1.0);
```



## ROUND - Realzahl als Integer runden _____ Funktion

Syntax: `ROUND(realausdruck);`

Die ROUND-Funktion wandelt eine Realzahl in eine Integerzahl um. Die Nachkommastellen werden korrekt gerundet.

Beispiele:
```
a:=ROUND(3.49999);
b:=ROUND(3.50001);
```

## SQR - Quadrat einer Zahl bilden _____ Funktion

Syntax:
```
SQR(integerausdruck);
SQR(realausdruck);
```

Die SQR-Funktion bildet das Quadrat des übergebenen Ausdrucks.

Beispiele:
```
Write(SQR(12),' ',12*12);
a:=SQR(2.718);
```

## SUCC - Nachfolger ermitteln _____ Funktion

Syntax:
```
SUCC(integerausdruck);
SUCC(zeichenausdruck);
```

Die SUCC-Funktion bildet den Nachfolger des Arguments. Bei Zahlen ist das beispielsweise 10001 von 10000, beim Zeichen 'y' ist es 'z'.

Beispiele:
```
a:=SUCC(10000);
a:=SUCC('y');
```

## TRUNC - Umwandlung einer Realzahl in Integer _____ Funktion

Syntax: `TRUNC(realausdruck);`

Die TRUNC-Funktion schneidet die Nachkommastellen einer Realzahl ab und wandelt sie in einen Integerwert.

Beispiele:
```
Write(TRUNC(4.99999));
Write(TRUNC(5.00001));
```

## UPCASE - Umwandlung in Großbuchstaben _____ Funktion

Syntax: `UPCASE(stringausdruck);`

Die UPCASE-Funktion wandelt alle Buchstaben im <stringausdruck> in Großbuchstaben um.

Beispiele:
```
Write(UPCASE('JRT-Pascal'));
x:=UPCASE(y);
```

## J. Externe Prozeduren

Externe Prozeduren sind eine Spezialität von JRT-Pascal. Diese sind in Diskettendateien im .INT-Format gespeichert. Einem Programm wird eine solche Prozedur durch eine formale Definition bekanntgemacht. Ruft das Programm dann eine dieser Prozeduren auf, wird sie von der Diskette nachgeladen und ausgeführt. Im Gegensatz zu Overlays bleibt sie aber weiterhin verfügbar und muß nicht ständig neu nachgeladen werden. Erst wenn der Speicherplatz knapp wird, führt EXEC eine Neuorganisation des Speichers durch und löscht die am längsten nicht mehr aufgerufenen externen Prozeduren. Dadurch wird wieder Platz frei, und andere Prozeduren können von der Diskette nachgeladen werden. EXEC sucht automatisch auf allen vorhandenen Diskettenlaufwerken. Das ist der Sinn der Anpassungsprozedur mit CUSTOMIZ.INT am Anfang!

Es werden bereits einige fertige externe Prozeduren mitgeliefert. Diese dienen mathematischen Berechnungen und für Diskettenoperationen:

- COS.INT Cosinus
- SIN.INT Sinus
- ARCTAN.INT Arcus-Tangens
- EXP.INT Exponentialfunktion
- LN.INT Natürlicher Logarithmus
- SQRT.INT Quadratwurzel
- ERASE.INT Löschen von Diskettendateien
- RENAME.INT Umbenennen von Diskettendateien

In Pascal-Programmen können Sie ganz einfach einen Verweis auf diese externen Prozeduren und Funktionen erstellen:

```
PROGRAM Demo;

VAR i:REAL;
FUNCTION SIN(x:REAL):REAL; EXTERN;

BEGIN
 WriteLn(SIN(90));
END.
```

Für die einzelnen Prozeduren sind die folgenden Deklarationen erforderlich:

```
FUNCTION Cos(x:REAL):REAL; EXTERN;
FUNCTION Sin(x:REAL):REAL; EXTERN;
FUNCTION ArcTan(x:REAL):REAL; EXTERN;
FUNCTION Exp(x:REAL):REAL; EXTERN;
FUNCTION Ln(x:REAL):REAL; EXTERN;
FUNCTION Sqrt(x:REAL):REAL; EXTERN;
PROCEDURE Erase(FileName:STRING[20]); EXTERN;
PROCEDURE Rename(OldName,NewName:STRING[20]; EXTERN;
```

Die Prozeduren RENAME und ERASE können beispielsweise so aufgerufen werden:

```
ERASE('C:DEMO.INT');
ERASE('PROG.COM');
RENAME('A:ALT.PAS','NEU.PAS');
RENAME(altdatei,neudatei);
```

Wenn Sie selbst externe Prozeduren schreiben wollen, ist das auch nicht sehr schwer. Sie können normale Prozeduren sehr einfach in externe umwandeln. Die Hauptarbeit besteht darin, vor den Prozedurkopf das Schlüsselwort EXTERN zu schreiben und an das Ende der gesamten Datei einen Punkt:



```
EXTERN

PROCEDURE Hallo;
BEGIN
 WriteLn('Hallo, hier ist die externe Prozedur!');
END; .
```

Die externe Prozedur wird ganz normal mit JPASCAL.COM compiliert. Externe Prozeduren und Funktionen können auch auf globale Variablen des Hauptprogramms zugreifen. Diese werden in der externen Datei dadurch bekanntgemacht, daß sie zwischen den Schlüsselworten EXTERN und PROCEDURE erneut deklariert werden:

```
EXTERN

CONST Pi = 3.141592653;
TYPE  Feld = ARRAY[1..40] OF INTEGER;
VAR   a: Feld;
      b: INTEGER;
      r: REAL;

PROCEDURE Hallo;
  WriteLn('b=',b,'r=',r);
END; .
```

Weder der Compiler noch EXEC.COM prüfen, ob die Parameterlisten bei den Deklarationen der externen Prozeduren in den externen Dateien und dem Hauptprogramm übereinstimmen. Das müssen Sie selbst sicherstellen!

Sie müssen im übrigen nicht für jede externe Prozedur eine eigene Datei anlegen. In einer Datei dürfen auch mehrere externe Funktionen und Prozeduren enthalten sein. Dann müssen Sie aber unbedingt beachten, daß die Referenzen im Hauptprogramm in derselben Reihenfolge stehen wie die Prozeduren in der externen Datei.

Wenn Sie ein Programm, das auf externe Prozeduren zugreift, fertiggestellt haben und dessen Laufzeit optimieren wollen, können Sie einige oder alle externen Prozeduren und Funktionen in das Hauptprogramm integrieren. Sie stehen dadurch viel schneller zur Verfügung, belegen allerdings auch ständig Platz im Speicher. Diese Integration übernimmt das Programm LINKER.INT.

Das Programm wird mit Unterstützung von EXEC geladen:

```
A>EXEC LINKER
** EXEC 2.0D **

Linker V2.0D

Name des Programms: DEMO

Haupt-Programmodul

DEMO          Modulumfang = 56 Bytes      Code = 33 Bytes

  1:SIN

Prozedurauswahl:
1 0

Externals werden bearbeitet

 Prozedur          Total     Umfang    Phase
 SIN                  33        444    1   2   3
         473
```

```
Bearbeitung ....
Ausgabemodul:     496 Bytes
Linking ist beendet
Programmende ...
A>
```

Was hier geschieht, ist folgendes: Der Linker zeigt alle externen Prozeduren an, die ein compiliertes Pascal-Programm benötigt. Hier ist es lediglich eine, nämlich SIN. Es sind aber bis zu 63 externe Prozeduren pro Programm zulässig. Diese werden vom Linker von 1 bis 63 durchnumeriert. Sie werden dann gefragt, welche Prozeduren ins Hauptprogramm integriert und welche weiterhin extern bleiben sollen. Dazu geben Sie die Nummern der einzubindenden Prozeduren an. Sobald Sie Ihre Eingabe beendet haben, tippen Sie <0>. Daraufhin arbeitet der Linker für einige Sekunden und erzeugt eine neue Version des Pascal-Programms. Die alte Version bleibt weiter bestehen und erhält auf der Diskette die Namenserweiterung .IN2.

Bei der Eingabe der Prozedurnummern besitzen die folgenden Werte eine besondere Bedeutung:

| | |
|---|---|
| 1 bis 63 | Prozedur zur Einbindung auswählen |
| -1 bis -63 | Prozedur wieder deselektieren |
| 100 | Alle Prozeduren einbinden |
| -100 | Alle Prozeduren wieder deselektieren |
| 0 | Auswahl beenden und Linker weiterarbeiten lassen |

Während der Arbeit von LINKER.INT müssen alle benannten externen Prozeduren auf der Arbeitsdiskette vorhanden sein.

## K. Compilerdirektiven

Bei der Suche nach Programmfehlern ist es oft recht praktisch zu wissen, welche Programmzeile und welche Prozedur gerade abgearbeitet wird. Dazu besitzt JRT-Pascal das Line- und Procedure-Tracing. Diese beiden Formen des Tracings müssen ausdrücklich aktiviert werden. Denn sie verlängern auch den Programmcode. Sie schalten das Tracing durch Einfügen der folgenden Compilerdirektiven ein und wieder aus:

| | |
|---|---|
| %LTRACE | Code für Zeilen-Trace erzeugen |
| %NOLTRACE | Codeerzeugung für Zeilen-Trace abschalten |
| %PTRACE | Code für Prozedur-Trace generieren |
| %NOPTRACE | Keinen Code für Prozedur-Trace generieren |

Die Bereiche, die vom Tracing erfaßt werden sollen, müssen mit SYSTEM-Aufrufen eingegrenzt werden:

| | |
|---|---|
| SYSTEM(LTRACE) | Zeilen-Trace aktivieren |
| SYSTEM(NOLTRACE) | Zeilen-Trace abschalten |
| SYSTEM(PTRACE) | Prozedur-Trace einschalten |
| SYSTEM(NOPTRACE) | Prozedur-Trace abschalten |

Aber die Direktive %LTRACE hat noch einen anderen Sinn: Tritt ein Laufzeitfehler auf, zeigt der Computer auch die Nummer der Zeile an, durch die der Fehler verursacht wurde. Und Sie können jederzeit eine Unterbrechung des Programms erreichen. Wenn Sie während des Programmablaufs Control-N drücken, gelangen Sie in den EXEC-Interrupt. Geben Sie ein Fragezeichen oder Leerzeichen ein, dann zeigt das Programm seine Möglichkeiten. Bei Verwendung von %LTRACE können Sie auch Control-A drücken und gelangen in den ACTIVAN-Interrupt **(Activity Analyzer)**. Hier können Sie untersuchen, welche Programmteile besonders häufig oder besonders selten aufgerufen werden und damit den Programmcode an den



passenden Stellen weiter optimieren. Eine Befehlsliste erhalten Sie durch Drücken der Leertaste.

Es werden von JRT-Pascal noch weitere SYSTEM-Aufrufe unterstützt:

| | |
|---|---|
| SYSTEM(CONS) | Ausgabe bei WRITE und WRITELN auf dem Bildschirm |
| SYSTEM(NOCONS) | Bildschirmausgabe ausschalten |
| SYSTEM(LIST) | Druckerausgabe einschalten |
| SYSTEM(NOLIST) | Druckerausgabe ausschalten |
| SYSTEM(WARNING) | %WARNING-Meldungen anzeigen |
| SYSTEM(NOWARNING) | %WARNING-Meldungen nicht anzeigen |
| SYSTEM(INITIALIZE) | Floppylaufwerke nach Diskettenwechsel zurücksetzen |
| SYSTEM(COMPRESS) | Garbage-Collection auslösen |

## L. Ein- und Ausgabe in JRT-Pascal

Dem Pascal-Standard entsprechend unterstützt JRT-Pascal die Bildschirmausgabe mit WRITE und WRITELN:

```
Write(Variable1,'Konstante',Variable2...);
WriteLn(Variable1,'Konstante',Variable2...);
```

Über die WRITE-Prozedur können auch Codes zur Bildschirmsteuerung ausgegeben werden:

```
PROCEDURE ClrScr;
BEGIN
  IF cpmplus THEN Write(CHR(27),'E',CHR(27),'H')
             ELSE Write(CHR(12));
END;
```

Die Formatierung von INTEGER- und REAL-Zahlen ist ebenfalls möglich:

```
Write(3.1415:12:3);
Write(33:4);
```

Diskettendateien können im TEXT- oder BINARY-Format verarbeitet werden, je nachdem, welche Datentypen in welcher Form gespeichert werden sollen. Programme, die Dateien im Klartext erzeugen wollen, verwenden das TEXT-Format. Geht es darum, auch Binärdateien oder .COM-Files zu lesen, ist BINARY angeraten, weil in diesen Dateien auch sonst reservierte Controlcodes vorhanden sind. Die Prozedur REWRITE übernimmt die Aufgabe, gleichzeitig eine Datei auf der Diskette zu schaffen und eine Dateivariable zu initialisieren. In anderen Pascal-Dialekten ist diese Aufgabe auf die Prozeduren ASSIGN und REWRITE aufgeteilt:

```
Rewrite(dateivariable,'dateiname',BINARY,puffergroesse);
Rewrite(dateivariable,'dateiname',TEXT,puffergroesse);
```

Die <dateivariable> bezieht sich auf eine FILE-Deklaration im VAR-Teil des Programms. Je größer der Wert für <puffergroesse> ist, desto weniger Diskettenzugriffe sind nötig. Sie sollten ein Vielfaches von 128 wählen. Empfehlenswert sind Größen wie 1024, 2048 oder 4096.

Mit WRITE und WRITELN können Sie dann sequentiell in eine geöffnete Datei schreiben:

```
Write(dateivariable;variable1,variable2...);
WriteLn(dateivariable;variable1,variable2...);
```

Eine bereits existierende Datei wird mit RESET zum Lesen geöffnet:

```
Reset(dateivariable,'dateiname',BINARY,puffergroesse);
Reset(dateivariable,'dateiname',TEXT,puffergroesse);
```

Sie können dann mit READ und READLN Daten lesen:

```
(* Tastatur *)  Read(variable1,variable2...);
                ReadLn(variable1,variable2...);

(* Dateien *)   Read(dateivariable;variable1,variable2...);
                ReadLn(dateivariable;variable1,variable2...);
```

Im Gegensatz zu anderen Pascal-Dialekten muß der <dateivariable> ein Strichpunkt folgen, nicht etwa ein Komma.

Die Funktionen EOF(dateivariable) und EOLN(dateivariable) werden dazu benutzt, das Ende einer Datei oder einer Zeile zu bestimmen. Eine mit den Prozeduren RESET und REWRITE geöffnete Datei wird mit CLOSE(dateivariable>) wieder geschlossen.

Die Standardprozeduren GET und PUT werden im Gegensatz zu manchen anderen Pascal-Implementationen unterstützt:

```
x:=dateivariable^; GET(dateivariable);
```
ist identisch mit `Read(dateivariable;x);`

```
dateivariable^:=x; PUT(dateivariable);
```
ist identisch mit `Write(dateivariable;x);`

JRT-Pascal unterstützt auch den Random-Zugriff auf Diskettendateien. Im Gegensatz zu sequentiellen Files kann hier auch ein Datensatz direkt angesteuert werden.

Direktzugriffsdateien werden mit OPEN geöffnet und mit der normalen CLOSE-Prozedur geschlossen. OPEN besitzt die folgende Syntax:

```
OPEN(dateivariable,'dateiname',BINARY);
OPEN(dateivariable,'dateiname',TEXT);
```

Existiert bereits eine passende Datei, so wird diese geöffnet. Sonst legt die Prozedur automatisch eine neue Datei an.

Normalerweise besitzen die Datensätze in Random-Dateien stets die gleiche Länge. In diesem Fall kann ein spezieller Datensatz mit Hilfe von RRN **(Relative Record Number)** angesteuert werden. Dazu geben Sie diese drei Buchstaben und darauf folgend die Nummer des Datensatzes bei READ oder WRITE an. Für den ersten Datensatz gilt RRN=0. Zwei Beispiele:

```
Write(dateivariable,RRN,26;variable1,variable2...);
Read(dateivariable,RRN,255;variable1,variable2...);
```

Die RRN-Nummer kann vom Typ INTEGER oder REAL sein. Wie groß die tatsächlichen Datensätze sind, ergibt sich aus der Deklaration der Datei. So ist die Satzlänge bei einem FILE OF REAL acht, weil jede REAL-Zahl aus acht Bytes besteht.

Die RRN-Methode besitzt den Nachteil, daß alle Datensätze gleich lang sein müssen. Deshalb kann auch RBA benutzt werden, die **Relative Byte Address.** Hier muß das Programm selbst die aktuelle Adresse eines Datensatzes ermitteln und übergibt der Laufzeitbibliothek eine Zahl, die den Abstand vom Dateianfang in Bytes angibt. So gilt für das erste Zeichen RBA=0. Der Aufruf erfolgt genau wie bei RRN:



```
Write(dateivariable,RBA,70000;variable1,variable2...);
```

## M. JRT-Pascal und Maschinensprache

In Programme von JRT-Pascal können ohne weiteres Portionen von Maschinencode eingebunden werden. Die Schnittstelle zur Maschinensprache kann auf verschiedenen Wegen erreicht werden.

### 1. Aufruf von BDOS- und BIOS-Routinen

JRT-Pascal besitzt eine eingebaute Prozedur mit dem Namen CALL. Sie übernimmt eine Sprungadresse und zwei Records mit den 8080/Z80-Registern. Über den BDOS-Einsprung an der Adresse 0005H oder durch Berechnung des BIOS-Beginns über den Warmsprungvektor in der Adresse 0001H können Routinen des CP/M-Betriebssystems aufgerufen werden.

Zur Registerübergabe hat sich der folgende Record bewährt:

```
TYPE Register = RECORD
                  CASE INTEGER OF
                    1: (Flag,A,C,B,E,D,L,H:CHAR);
                    2: (AF,BC,DE,HL:INTEGER);
                  END;
```

Mit Hilfe dieses varianten Records können Sie die Z80-Register sowohl mit acht als auch sechzehn Bit Breite ansprechen:

```
Register.A:=CHR(32);
Register.HL:=3FACH;
Register.AF:=16324;
```

Während es sich also bei den 8-Bit-Registern um CHARs handelt, sind es bei den Doppelregistern INTEGER-Zahlen.

Als nächstes definieren Sie zwei Records, einen für die Übergabeparameter, einen für die Registerwerte, die nach dem Prozeduraufruf ausgelesen werden sollen:

```
VAR PutReg,GetReg:Register;
```

Sie können dann eine BDOS-Funktion so aufrufen:

```
CALL(5,PutReg,GetReg);
```

Geht es beispielsweise darum, einen Disketten-Reset auszulösen, würde das in Maschinensprache so geschrieben:

```
8080-Code:      Z80-Code:
MVI  C,13       LD   C,13
CALL 5          CALL 5
```

Mit JRT-Pascal hieße dies so:

```
PROCEDURE DiskReset;
BEGIN
  PutReg.C:=CHR(13);
  CALL(5,PutReg,GetReg);
END;
```

Zur Ermittlung der Versionsnummer des verwendeten CP/M können Sie so verfahren:

```
PROCEDURE GetVersion;
BEGIN
  PutReg.C:=CHR(12);
  CALL(5,PutReg,GetReg);
  IF GetReg.H=CHR(0) THEN Write('CP/M ') ELSE Write('MP/M ');
  Write(HEX$(GetReg.L));
END;
```

## 2. PEEK und POKE in Pascal

Zum Auslesen und Schreiben von Werten in einzelne Speicherzellen können Sie die eingebaute MAP-Prozedur verwenden. Sie arbeitet mit Zeigervariablen und rechnet grundsätzlich mit 16-Bit-Werten:

```
FUNCTION BiosAddr:INTEGER;

(* BIOS-Adresse ermitteln ueber Warmstartvektor in 0001H *)

VAR Zeiger:^INTEGER;

BEGIN
  MAP(Zeiger,0001H);
  BiosAddr:=(Zeiger^-3);
END;
```

Über den umgekehrten Weg ist auch das Schreiben von Werten möglich:

```
MAP(Zeiger,0030H); Zeiger^:=2710H;
```

Ein einfacher hexadezimaler Speicherauszug wird mit diesem kleinen Programm auf dem Bildschirm dargestellt:

```
PROGRAM dump;

VAR i:INTEGER;
    Zeiger:^INTEGER;

BEGIN
 FOR i:=0 TO 30 DO BEGIN
     Map(Zeiger,i);
     Write(Hex$(Zeiger^),' ');
     END;
END.
```

Die Prozedur MAP ähnelt der dynamischen Variablenverwaltung durch NEW insofern, daß bei jedem MAP-Aufruf Speicherplatz in der Zeigertabelle belegt wird. Deshalb sollten Sie möglichst nach dem Auslesen eines Werts den Platz mit DISPOSE wieder freigeben:

```
DISPOSE(Zeiger);
```

## 3. Der Assembler JASSEM.INT

Wenn Sie selbst kleine oder größere Maschinenprogramme erstellen wollen, können Sie den mitgelieferten 8080-Assembler JASSEM.INT benutzen. Er ist kompatibel zum Standardassembler ASM.COM, der zu CP/M 2.2 gehört.



Dazu erstellen Sie zuerst mit dem Texteditor den Assembler-Quellcode und lassen dann den Assembler eine Datei im .INT-Zwischenformat erzeugen. Der Aufruf von Assembler-Prozeduren erfolgt dann von Programmen aus, als wären sie normale externe Prozeduren.

Jedem Assemblerprogramm muß ein fünf Bytes langer Code vorangestellt werden, damit EXEC erkennt, daß es sich um keine Pascal-Prozedur handelt. Ein kurzes Maschinenprogramm zur Ausgabe eines Textes auf dem Bildschirm sieht so aus:

```
CODE    DB    95,6,0,92,0      ; Es folgt Assemblercode
START   MVI   C,9              ; BDOS-9/Stringausgabe
        LXI   D,TEXT           ; Zeiger auf Text
        CALL  5                ; Aufruf des BDOS
        RET                    ; Ruecksprung nach Pascal
TEXT    DB    'Ein kleines Beispiel fuer JASSEM$'
        END
```

Im Programm dürfen alle Registerinhalte außer SP beliebig verändert werden. Assemblerprozeduren müssen stets mit RET abgeschlossen werden.

Wenn Sie dieses Programm TEXTOUT.ASM nennen, können Sie es im Hauptprogramm so als externe Prozedur deklarieren:

```
PROCEDURE TextOut; EXTERN;
BEGIN
 TextOut;
END.
```

Da der Geschwindigkeitsvorteil von Maschinencode aber durch das notwendige Nachladen von der Diskette zunichte gemacht würde, werden Sie solche Assemblerroutinen bald mit dem Programmlinker LINKER.INT fest in das Hauptprogramm einbinden.

Der Assembler wird mit EXEC aufgerufen:

```
A>EXEC JASSEM

** EXEC 2.0D **

JRT-Assembler V2.0D
Copyright 1982 JRT Systems

Programmname: TEXTOUT
Option: C=COM 1=Pass1 N=Keine: N

0000 5F06005C00 1: CODE  DB    95,6,0,92,0 ; Es folgt Assemblercode
0005 0E09       2: START MVI   C,9          ; BDOS-9/Stringausgabe
0007 110E00     3:       LXI   D,TEXT       ; Zeiger auf Text
000A CD0500     4:       CALL  5            ; Aufruf des BDOS
000D C9         5:       RET                ; Ruecksprung nach Pascal
000E 45696E2042 6: TEXT  DB    'Ein kleines Beispiel fuer JASSEM$'
0027            7:       END

     Symbol         Flag    Wert
  1: CODE             40       0  0000
  2: START            40       5  0005
  3: TEXT             40      14  000E

 0 Fehler entdeckt
Assembler fertig: TEXTOUT
Programmende ...
```

Die Option <C> bedeutet, daß der Assembler ganz normale CP/M-Programme mit der Namenserweiterung .COM erzeugen soll. Daher kann er vollständig ASM.COM ersetzen. Führen Sie mit <1> nur den ersten Pass der Assemblierung durch, durchsucht der Assembler den Quellcode lediglich auf Fehler und erzeugt ein Bildschirm-Listing. Er erstellt aber keine Objektdatei.

Folgende Assemblerdirektiven werden unterstützt:

| | |
|---|---|
| ORG | Adreßzähler festlegen (bei externen Prozeduren nicht benutzt)<br>Beispiel: `ORG 0100H` |
| EQU | Konstante festlegen<br>Beispiel: `BDOS EQU 0005H` |
| SET | Wie EQU, kann aber später neu definiert werden<br>Beispiel: `B SET 25` |
| DB | Bytekombination<br>Beispiel: `DB 'A',3,4,255` |
| DW | Speicherworte<br>Beispiel: `DW 3FACH,16384,255,32767` |
| DS | Reservierung von Speicherplatz<br>Beispiel: `DS 255` |
| READ | Definiert wie SET, der Wert wird aber während der Assemblierung von der Tastatur eingelesen.<br>Beispiel: `xyz READ` |
| WRITE | Ein Text wird während der Assemblierung ausgegeben.<br>Beispiel: `WRITE 'Bitte geben Sie einen Wert ein'` |

Außerdem können Sie mit IF, ENDIF und ELSE eine bedingte Assemblierung von Programmteilen erreichen.

Der Assembler verarbeitet auch mathematische Ausdrücke von Integerwerten. Es werden die vier Grundrechenarten unterstützt: * / + -. Außerdem dürfen die Logikoperatoren NOT, AND, OR und XOR sowie die Operatoren MOD (Divisionsrest), HIGH (Highbyte eines 16-Bit-Werts) und LOW (Lowbyte eines 16-Bit-Werts) benutzt werden. Klammern sind zur Schachtelung erlaubt. Vergleiche sind mit den Kurzbezeichnern EQ, NE, LT, LE, GT und GE zu programmieren:

```
EQ    equal              gleich
NE    not equal          ungleich
LT    less than          kleiner als
LE    less or equal      kleiner oder gleich
GT    greater than       größer als
GE    greater or equal   größer oder gleich
```

Fehler und Probleme: Lange Kommentarzeilen können bei JASSEM.INT zu einer mysteriösen Fehlermeldung führen. Verwenden Sie also möglichst kurze Kommentare! Leere Zeilen beantwortet der Assembler mit einer Warnung. Setzen Sie stattdessen einen Strichpunkt ein, der dann einen leeren Kommentar markiert.

Sie können zwischen Pascal-Programmen und Assemblercode Daten austauschen. Das erfordert aber einiges an Programmierkönnen und sollte nicht von Anfängern versucht werden. In einfachen Fällen reicht es im allgemeinen aus, Werte mit der Prozedur MAP in einen unbenutzten Teil des Speichers zu schreiben und von der Assemblerprozedur auslesen zu lassen. Das ist zwar nicht elegant, aber einfach zu programmieren.



Sie können dem Assemblerprogramm aber auch direkt Werte wie einer ganz normalen Prozedur oder Funktion übergeben. JRT-Pascal unterhält für diesen Zweck einen speziellen Datenstack. Der Datenstack enthält alle statischen Variablen, Parameter, Rückgabewerte von Funktionen und Linkage-Blocks von Prozeduren. Es werden drei Zeiger verwaltet, um auf den Datenstack zuzugreifen. Beim Aufruf einer externen Maschinenroutine sind diese in den Registern enthalten:

HL zeigt auf BASE, die untere Adresse des Datenstacks.
DE zeigt auf CUR, den aktuellen Linkage-Block der Prozedur.
BC zeigt auf TOS, also hinter das letzte belegte Byte des Stacks.

Es gibt zwei prinzipiell verschiedene Wege zur Übergabe von Daten. Der eine beruht auf der Tatsache, daß der Datenstack die im Pascal-Hauptprogramm deklarierten globalen Variablen der Reihe nach enthält. Bei einer Deklaration wie VAR i,j:INTEGER sind die Werte der Variablen i und j der Reihe nach auf dem Datenstack abgelegt. Um auf die globalen Variablen zuzugreifen, addieren Sie den konstanten Wert 6 zu BASE. Durch Auslesen der so ermittelten Speicheradresse erreichen Sie die zuerst deklarierte globale Variable. Abhängig von der Bytelänge des entsprechenden Variablentyps können Sie auf die weiteren globalen Variablen zugreifen.

Datenstack bei Aufruf BEISPIEL('Z',5):

```
'Z'  5     Bytelänge  Linkage-Block
5A   05 00 03 00      xx xx xx xx xx xx   yy
                       |                  |
                       CUR                TOS
```

Datenstack bei Aufruf x:=BEISPIEL(5,9):

```
      5     9     Bytelänge  Linkage-Block
zzzz  05 00 09 00 04 00      xx xx xx xx xx xx  yy
|                            |                  |
Rückgabewert                 CUR                TOS
```

Der angegebene Linkage-Block wird von JRT-Pascal zur internen Verwaltung benötigt und hat für den Programmierer keine Funktion.

Bei Werteparametern wird der Parameter in seiner ganzen Länge auf dem Datenstack gespeichert. Bei Variablenparametern hingegen enthält der Datenstack nur die Speicheradresse des Parameters.

Einige Assembler-Beispielprogramme finden Sie auf der Diskette. Es handelt sich um SETBIT, RESETBIT und TESTBIT. Diese verwenden ausschließlich den zweiten Typ der Parameterübergabe. Damit Sie mit der Parameterübergabe experimentieren können, finden Sie im folgenden ein Assemblerprogramm, das mit JASSEM.INT assembliert und als externe Prozedur oder Funktion in Programme eingebunden wird. Es zeigt den Zustand des Datenstacks in hexadezimaler Form an:

```
BDOS    equ  0005h
STROUT  equ  9
CODE    db   95,6,0,92,0
        jmp  TEST
str1    db   13,10,10,'BASE HL: $'
str2    db   13,10,   'CUR  DE: $'
str3    db   13,10,   'TOS  BC: $'
crlf    db   13,10,10,'$'
TEST    push b ! push d ! push h
        push b ! push d ! push h
msg1    mvi  c,STROUT              ; Stringausgabe
        lxi  d,str1                ; Zeiger auf Meldung 1
```

```
        call BDOS                   ; Meldung ausgeben
        pop  h                      ; HL wiederherstellen
        mov  a,h ! call hexout
        mov  a,l ! call hexout
msg2    mvi  c,STROUT               ; Stringausgabe
        lxi  d,str2                 ; Zeiger auf Meldung 2
        call BDOS                   ; Meldung ausgeben
        pop  d                      ; DE wiederherstellen
        mov  a,d ! call hexout
        mov  a,e ! call hexout
msg3    mvi  c,STROUT               ; Stringausgabe
        lxi  d,str3                 ; Zeiger auf Meldung 3
        call BDOS                   ; Meldung ausgeben
        pop  b                      ; BC wiederherstellen
        mov  a,b ! call hexout
        mov  a,c ! call hexout
creturn mvi  c,STROUT               ; Stringausgabe
        lxi  d,crlf                 ; Zeiger auf CR und LF
        call BDOS                   ; ausgeben
        pop h!pop d!pop b!push h
        mvi  b,26                   ; 26 Hexziffern auszugeben
loop1   mov  a,l ! call hexout
        mvi  a,' ' ! call display
        inx h                       ; Zeiger erhoehen
        dcr b                       ; Zaehler vermindern
        jnz loop1                   ; Wiederholen bis fertig
        pop h                       ; HL wiederherstellen
        mvi a,13 ! call display
        mvi a,10 ! call display
        mvi b,26                    ; 26 Werte auszugeben
loop2   mov a,m ! call hexout
        mvi a,' ' ! call display
        inx h                       ; Zeiger erhoehen
        dcr b                       ; Zaehler vermindern
        jnz loop2                   ; Weitermachen bis fertig
        ret                         ; Ruecksprung nach Pascal
hexout  push psw                    ; Hexziffern-Ausgabe
        rar ! rar ! rar ! rar
        call ascii
        pop psw
ascii   ani 0fh
        adi '0'
        cpi ':' ! jc display
        adi 7
display push b ! push d ! push h    ; Bildschirmausgabe
        mov  e,a ! mvi  c,2
        call BDOS
        pop  h ! pop d ! pop b
        ret
        end
```

Je nachdem, wie Sie dieses Assemblerprogramm im Pascal-Code deklarieren, zeigt es unterschiedliche Werte für den Datenstack an. Einige Beispiele:

```
PROCEDURE testarg; EXTERN;
PROCEDURE testarg(a:INTEGER); EXTERN;
PROCEDURE testarg(a:CHAR,v:INTEGER); EXTERN;
FUNCTION testarg(x:REAL):INTEGER; EXTERN;
```

Ja, Sie haben richtig gesehen: Ohne Änderungen im Assemblercode können Sie eine externe Assemblerfunktion auch als Prozedur deklarieren. Dann geht eben der Rückgabewert verloren.



## N. Fehlermeldungen

Die angepaßte Version von JRT-Pascal besitzt deutschsprachige Fehlermeldungen. Wohl kein anderer Pascal-Compiler kennt so umfangreiche Fehlermeldungen.

Wenn im Laufzeitsystem ein Fehler auftritt, erhalten Sie die folgende Anzeige:

```
Addr:  xxxx   Prog:  xxxx   Size:  xxxx
Base:  xxxx   Cur:   xxxx   Tos:   xxxx
Low:   xxxx   Compr: xxxx   Purge: xxxx
```

`Addr:` Speicheradresse, an der der Fehler auftrat
`Size:` Größe des Hauptprogramms
`Base:` Unterste Adresse des Datenstacks
`Cur:` Adresse des aktuellen Aktivierungsblocks der Prozedur
`Tos:` Adresse hinter dem Ende des Datenstacks
`Low:` Unterste Adresse eines dynamischen Speicherblocks
`Compr:` Zahl der bisherigen automatischen Garbage-Collections
`Purge:` Zahl der aus Speicherplatzgründen gelöschten externen Prozeduren

Oft wird auch der Name der zuletzt geöffneten Datei ausgegeben. Ein spezielles binär codiertes Flagbyte teilt weitere Informationen mit:

`Bit 80H` Datei ist geöffnet
`Bit 40H` Direktzugriff
`Bit 20H` `TEXT`-Modus
`Bit 10H` `EOLN` erreicht
`Bit 08H` Datei zum Lesen geöffnet
`Bit 04H` `EOF` erreicht

Außerdem erhalten Sie eine Liste der externen Prozeduren, die ebenfalls Flagbytes enthält:

`Bit 80H` Externe Prozedur ist gerade aktiv
`Bit 40H` Externe Prozedur durch `LINKER.INT` mit Programm verbunden
`Bit 20H` Externe Prozedur ist gerade im Speicher
`Bit 10H` Datei-Kontrollblock (FCB) der Prozedur ist geöffnet
`Bit 04H` Adresse der Prozedur ist real, nicht virtuell

JRT-Pascal kennt die folgenden Fehlermeldungen:

**"Erwartet"-Meldungen:**

PROGRAM EXTERN END BEGIN DO OF THEN TO/DOWNTO UNTIL
= .. : [ ] ( ) ^ :=
Programmidentifier Semikolon Identifier Komma Typenidentifier Einfacher Typ RECORD-Identifier Zeigertyp File-Identifier Integer (vorzeichenlos) BOOLEAN-Ausdruck Integerausdruck REAL-Ausdruck Externe Prozedur/Funktion String- oder Zeichenausdruck Stringvariable

**Sonstige Fehlermeldungen:**

Ungueltiger Pseudocode erzeugt, Systemfehler
Uebersetzung endet nicht mit Punkt
Undefin. Label
Konstante falsch
Syntaxfehler: Konstante
Unterbereichkonstanten sind nicht kompatibel
Unterbereich: erste Konstante groesser als zweite

Undefin. Typenidentifier
Mehr als acht Felddimensionen deklariert
Mehr als 10 Dateien (... deklariert)
Syntaxfehler bei WITH
Undeklarierte Variable
Undeklariertes Datenfeld
Syntaxfehler: Felddeklaration
Ungueltiger Ausdruckstyp bei Feldreferenz
Ungueltige Feldreferenz
Ungueltiger Ausdruckstyp: Parameter
Parameter-Syntaxfehler
Falsche Parameterzahl
Inkompatible SET-Typen
Inkompatible Typen in SET-Subrange (SET-Unterbereich)
Ungueltiger SET-Typ
Ungueltiger Parameter bei Systemprozedur
Inkompatible Datentypen im Ausdruck
Inkompatible Datentypen bei Zuweisungsbefehl
Inkompatible Datentypen bei CASE
Datentypen falsch bei FOR
Datentyp falsch bei Ausdruck
Syntaxfehler bei Ein-/Ausgabebefehl
RBA/RRN-Ausdruck kein Integertyp
Dateiname falsch bei OPEN/RESET/REWRITE
Puffergroesse-Parameter kein Integer
Parameter falsch bei SYSTEM
'Factor'-Syntaxfehler
Erster Operand beim SET-Test hat ungueltigen Typ
Syntax der Parameterliste
Parameterfehler in OPEN/RESET/REWRITE
ABS-Parameter nicht REAL oder INTEGER
Erster Parameter bei COPY kein STRING-Typ
Zweiter Parameter bei COPY kein INTEGER-Typ
Dritter Parameter bei COPY kein INTEGER-Typ
Erster Parameter bei POS kein STRING-Typ
Zweiter Parameter bei POS kein STRING-Typ
Dritter Parameter bei POS kein INTEGER-Typ
Nur eine (genau eine) Stringvariable bei READ erlaubt
Blockstruktur des Programms falsch
Feldindex zu gross
Feldindex zu klein
Zeigervariable hat falsche Indexadresse
Systemfehler bei Speicherplatzsuche/Kompression
Systemfehler bei Tabellen der dyn. Speicherung, eventuell wegen Speicheroverlay
Speichertabellen voll, mehr als 1632 Blocks allokiert
Zu wenig dynam. Speicher
Ungueltiger Parameter bei SYSFUN-Funktion
Integerdivision durch 0; wenn Dividend > 0 Quotient = +32767, sonst -32768. Der Rest ist immer 0
Systemfehler im Monitor
Zu wenig Speicher fuer Datenstack
Ungueltiger Pseudocode entdeckt, eventuell wegen Speicheroverlay oder inkorrekt definierter externer Prozedur
Fehler bei Oeffnen (.. der Datei) (Random-Zugriff)
Fehler beim Zugriff auf Random-File
Ungueltiges Datenformat bei E/A-Befehl
Falsches Datenformat bei Eingabebefehl
Zugriff auf ungeoeffnete Datei
Daten ueberschreiten Recordgroesse in einem RRN-Schreibbefehl
Fehler beim Oeffnen: Datei existiert nicht -- Directory voll
Fehler bei sequent. Schreiben
Random-Zugriff auf sequent. File



Sequent. Zugriff auf Random-File
Formatfehler bei Ein/Ausgabe
Ungueltiger Parameter bei der SYSTEM-Prozedur
Falscher TRUNC/ROUND-Wertebereich; wenn Parameter positiv, ist das Resultat 32767,sonst -32768
Fliesskommadivision durch Null; wenn Dividend positiv, wird Quotient +.99999999999999e+63, sonst -.9999999999999e+63
Fehler beim Laden externer Prozeduren
INT-Datei fuer ext. Prozedur nicht da
Falscher Code an EXEC uebergeben
Fehler bei COPY, Startposition + Laenge - 1 groesser als Stringlaenge
Fehler bei POS, Startposition groesser als Stringlaenge
Falsches Datenformat bei Ein-/Ausgabebefehl
Fehler: Fileerweiterung
Diskette ist voll
Directory/Verzeichnis ist voll
Random-Zugriff nicht unterstuetzt
Fehler bei RESET, Datei nicht da
Libraries mit ext. Prozeduren n.unterstuetzt
Zugriff auf Zeichen in dynamischem String: Index groesser als Laenge
Bei DELETE-Prozedur ist Positionsparameter < 1
Bei DELETE sind Position und Laenge groesser als die aktuelle Stringlaenge
Bei DELETE ist die Laengenangabe kleiner als 0
Bei INSERT ist die Positionsangabe kleiner als 1
Bei INSERT ist die Positionsangabe groesser als die Stringlaenge
Bei INSERT wuerden die Parameter bewirken, dass die maximale Stringlaenge ueberschritten wird
Falsche dynamische Speicheradresse, evtl. wegen falscher VAR/TYPE-Deklaration
Ueber-/Unterlauf bei Fliesskomma-Multiplikation/Division/Addition/Subtraktion