

R 65 LEVEL I PASCAL

=====
PASCAL Compiler Version 5E)

Das R65 LEVEL I PASCAL ist eine Implementation der Programmiersprache PASCAL fuer das R65 Mikrocomputer-System. Es handelt sich dabei um eine Auswahl der jenigen Eigenschaften von PASCAL, die fuer den Einsatz von Mikroprozessoren zur Steuerung und zur Datenverarbeitung wesentlich sind. Dabei wurde bewusst auf einige wesentliche Eigenschaften von PASCAL verzichtet. Dies erlaubt auch, das ganze R65 LEVEL I PASCAL SYSTEM mit 24 K RAM und einer Minifloppy-Disk zu betreiben.

Kap. 1: DATENTYPEN

Die im folgenden behandelten einfachen Datentypen werden vom Compiler unterstuetzt. Die Definition von anderen Datentypen (wie etwa REAL oder SUBRANGE) ist vorlaufig nicht moeglich.

1.1 Der Datentyp "INTEGER": (I)

(Da kein Typ BOOLEAN definiert ist, ist dieser in INTEGER enthalten)

Bereich der ganzen Zahlen von -32768 bis +32767.

Operationen: + - * DIV (Arithmetisch)
 AND OR EOR NOT (Bitweise logisch)
 SHL SHR (Shift-Operationen)
 > < >= <= = <> (Vergleich)

Funktionen: ORD(X: CHAR): INTEGER (ASCII-Wert)
 CHR(X: INTEGER): CHAR (Buchstabe)
 ODD(X: INTEGER): INTEGER (Ungerade)
 MEM(X: INTEGER): INTEGER (Peek-Funktion)

Konstanten: 1,2,3,-1,-2,.....
 \$1,...,\$F00,.....

In der R65 SYSLIB zusaetzzlich definiert:

Konstanten: MAXINT, TRUE, FALSE
 sowie systemspezifische Adressen

Funktionen: ABS(X: INTEGER): INTEGER
 MOD(X,Y: INTEGER): INTEGER

sowie systemspezifische Funktionen

Der Typ BOOLEAN wird durch die Werte TRUE=1 und FALSE=0 repräsentiert, ohne explizit einführt zu werden.

1.2 Der Datentyp CHAR: (C)

- - - - -

Der Datentyp CHAR bezeichnet einen mit 8 Bits dargestellten Buchstaben (7-Bit ASCII Code, Bit 8 = 0).

Operationen: < > <= >= <> = (Resultat INTEGER)

Funktionen: CHR(X: INTEGER): CHAR
ORD(X: CHAR): INTEGER
LOW(X: PACKED CHAR): CHAR
HIGH(X: PACKED CHAR): CHAR
PACKED(X, Y: CHAR): PACKED CHAR

Konstanten: 'A', 'B',
CHR(1), (Nicht-printbare Zeichen)

In der R65 SYSLIB zusätzlich definiert:

Konstanten: BELL, BL, CDM, CLE, CSC, CLN, CR, CRI, CUP, DCH,
DLN, DSC, EOF, ESC, HOM, ICH, IHM, ILM, IUD,
LF, NUD, POF, PON, RDN, RUP

Die interne Speicherung einer Variablen vom Typ CHAR braucht ebenfalls 16 Bits, wie alle anderen Typen.

1.3. Der Datentyp PACKED CHAR (P)

- - - - -

Der Datentyp PACKED CHAR bezeichnet zwei Buchstaben, die mit einem 16-Bit Wort dargestellt werden.

Operationen: < > <= >= <> = (Resultat INTEGER)

Funktionen: LOW(X: PACKED CHAR): CHAR
HIGH(X: PACKED CHAR): CHAR
PACKED(X, Y: CHAR): PACKED CHAR

Konstanten: 'AA', 'AB',

Dieser Datentyp wurde speziell für die 16-Bit PASCAL Darstellung einführt, er ist im Standard-PASCAL nicht enthalten.

1.4 Der Datentyp FILE (F)

- - - - -

Der Datentyp File wird gebraucht, um mit sequentiellen TEXT-Files zu arbeiten. Die Behandlung dieser Files erfolgt nicht gleich, wie im Standard-PASCAL. Im R65 PASCAL wird mit dem Datentyp FILE der Name des Textfiles bezeichnet. Vom File werden oder auf das File beschrieben werden können hingegen Werte vom Typ CHAR, PACKED CHAR und INTEGER.

Operationen: < > <= >= <> = (Resultat Integer)

Funktionen: Keine

Konstanten: @0, @1,

In der R65 SYSLIB zusätzlich definiert:

Konstanten:	INPUT	(Gebufferte Keyboard-Eingabe)
	KEY	(Ungebufferte Keyboard-Eingabe)
	OUTPUT	(Video-Display)
	PRINTER	(Hard Copy Output)

1.5 Tabelle der Operationen und Funktionen

Funktionsname	Argument-Typ	Resultat-Typ
CHR(I1)	INTEGER	CHAR
HIGH(P1)	PACKED CHAR	CHAR
LOW(P1)	PACKED CHAR	CHAR
MEM(I1)	INTEGER	INTEGER
NOT(I1)	INTEGER	INTEGER
ODD(I1)	INTEGER	INTEGER
ORD(C1)	CHAR	INTEGER
PACKED(C1,C2)	CHAR	PACKED CHAR
(ABS(I1)	INTEGER	INTEGER)
(MOD(I1,I2)	INTEGER	INTEGER)
()	Jeder Typ	Gleicher wie Argument
*	INTEGER	INTEGER
DIU	INTEGER	INTEGER
AND	INTEGER	INTEGER
SHL	INTEGER	INTEGER
SHR	INTEGER	INTEGER
+	INTEGER	INTEGER
-	INTEGER	INTEGER
OR	INTEGER	INTEGER
XOR	INTEGER	INTEGER
<	Jeder Typ	INTEGER
>	Jeder Typ	INTEGER

<=	Jeder Typ	INTEGER
>=	Jeder Typ	INTEGER
<>	Jeder Typ	INTEGER
=	Jeder Typ	INTEGER

(>) bedeutet, dass diese Funktionen in der R65 SYSLIB definiert sind. Die Einteilung in Gruppen zeist auch die Prioritaetseinteilung bei der Aufloesung von arithmetischen Ausdruecken.

Der Einsatz von Funktionen und Operationen auf falsche Datentypen resultiert in einem TYPE MISSMATCH Fehler im Compiler. Dieser Test laesst sich allerdings bei der Uebersabe von Parametern an Proceduren und Funktionen unterdruecken (Siehe unten).

Kap. 2: DATENSTRUKTUREN

Datenstrukturen beschreiben die Anordnung von Daten der verschiedenen Typen. Im Standard-PASCAL sind eine ganze Reihe von Datenstrukturen erlaubt. Im R65 PASCAL sind jedoch nur eine kleine Auswahl zu finden.

2.1. Die Konstante (CONST) (C)

Konstanten werden am Anfang eines BLOCKS mit einer Konstantendeklaration deklariert:

```
CONST A=5; NONSENSE=$FFFF;
      INPUT=@0; CR=CHR(13);
```

Konstanten werden durch den Compiler berechnet und dann jeweils bei Gebrauch direkt mit LOAD IMMEDIATE Codes eingesetzt. Konstanten duerfen deshalb nicht durch eine Zuweisung geaendert werden.

2.2 Die Variable (VAR) (U)

Variablen muessen in einer Variablen-Deklaration deklariert werden:

```
VAR A,B : INTEGER;
     CH1,CH2 : CHAR;
     DEVICE : FILE;
     TOKEN : PACKED CHAR;
```

Variablen kann ein Wert (vom gleichen Typ!) zugeordnet werden und die Werte koennen in Ausdruecken verwendet

werden:

```
A      :=  B + 5;
CH1    :=  CHR(A);
TOKEN  :=  PACKED(CH1,CH2);
DEVICE :=  OUTPUT;
```

Jede Variable wird durch ein WORT (16 Bits) auf dem PASCAL RUNTIME STACK dargestellt.

2.3 Der ARRAY

(A)

Der Array ist eine lineare Anordnung von einfachen Daten, die durch einen INDEX adressiert werden koennen. Vorlaeufig sind nur eindimensionale Arrays mit einem INDEX, der von 0 startet, erlaubt. Arrays muessen deklariert werden. Die Indizes von Arrays werden vorlaeufig nicht ueberprueft!

```
VAR   TABLE:  ARRAY[5] OF CHAR;
      VALUES: ARRAY[CONSTANT] OF INTEGER;
      DEVICES: ARRAY[3] OF FILE;
```

Array-Elemente koennen gleich wie Variablen eingesetzt werden. Sie lassen sich auch als KONSTANTE PARAMETER, nicht aber als VARIABLE PARAMETER an Prozeduren uebergeben. Die Uebersaabe eines Arrays als sanzes ist nicht moeglich.

2.4 Der KONSTANTE PARAMETER

(D)

Parameter werden an der Nahtstelle von Prozeduren und Funktionen eingesetzt. KONSTANTE PARAMETER erhalten beim Aufruf der Prozedur einen Wert (Beim Aufruf kann ein Ausdruck des entsprechenden Typs eingesetzt werden). Innerhalb der Prozedur koennen KONSTANTE PARAMETER nicht mehr abgeaendert werden.

```
PROC PRINTNUM (DU: DEVICE; NUM: INTEGER);
BEGIN
  WRITE(@DU, '%', NUM, CR, LF)
END;
```

Der Test auf korrekte Typen kann unterbunden werden, falls dies fuer eine Procedur, die verschiedene Datentypen bearbeiten soll, notwendig ist. Diese Unterbindung durch das Zeichen % gilt nur fuer die Uebergabe, innerhalb der Procedur gelten weiterhin alle Regeln.

```

PROC PUSH (X: %INTEGER); (PUSH any value onto
                           a user-stack)
BEGIN
  S:=S+1; IF S>MAX THEN ERROR;
  STACK[S]:=X
END;

```

Mit dieser Prozedur kann nun jeder Datentyp als INTEGER auf STACK abgespeichert werden:

```

PUSH(CH1);
PUSH(A);
PUSH(DEVICE);

```

Es ist klar, dass diese ungeprüfte Übergabe von Werten nur eingesetzt werden sollte, wenn dies unbedingt notwendig ist, da normalerweise die Typenüberprüfung wesentlich zur Sicherheit eines PASCAL Programms beiträgt.

2.5 Der VARIABLE PARAMETER (W)

Der VARIABLE PARAMETER wird eingesetzt, wenn eine Prozedur nicht nur einen INPUT benötigt, sondern auch einen OUTPUT erzeugt. Dabei wird nun an die Prozedur nicht ein Wert, sondern eine VARIABLE uebergeben. Dies bedeutet, dass die interne Variable (in der Prozedur) beim Aufruf den Wert der externen Variablen (im Aufruf) erhält. Der Wert dieser internen Variablen kann nun abgeändert werden. Am Ende der Prozedur wird nun der Wert der internen Variablen wieder auf die externe Variable uebertragen.

Beispiel (Vektoraddition):

```

PROC VECTORADD
  (CONST X1,X2,Y1,Y2: INTEGER; VAR Z1,Z2: INTEGER);
BEGIN
  Z1:=X1+Y1; Z2:=X2+Y2
END;

```

Betrachten wir nun den folgenden Aufruf:

```

U1:=3; U2:=5;
W1:=10; W2:=20;
VECTORADD(U1,U2,W1,W2,R1,R2);
WRITE(R1,' ',R2);

```

Durch diesen Aufruf erhält nun R1 den Wert 13 und

R2 den Wert 25, waerend die Prozedur VECTORADD die Werte von U1,U2,W1 und W2 nicht aendern kann.
Gleich wie bei konstanten Parametern ist auch bei VARIABLEN PARAMETERN die Unterdrueckung der Typenueberpruefung moeslich:

```
PROC POP(VAR X: XINTEGER);
BEGIN
  IF S<0 THEN ERROR;
  X:=STACK[S]; S:=S-1
END;
```

Beachte, dass bei der Definition von Prozeduren CONST default ist, d.h. wenn nicht explizit VAR beschrieben ist, werden KONSTANTE PARAMETER angenommen. Ebenfalls ist der Typ INTEGER default und muss nicht explizit beschrieben werden.

2.6 Funktionen

(F)

Eine Funktion ist eine besondere Art einer Datenstruktur. Das Resultat einer Funktion kann ebenfalls jeden beliebigen Typ haben:

```
FUNC ABS(X: INTEGER): INTEGER;
BEGIN
  IF X<0 THEN ABS:=-X ELSE ABS:=X
END;

FUNC BIGLETTER(CH1: CHAR): CHAR;
BEGIN
  IF (CH1>='a') AND (CH1<='z')
  THEN BIGLETTER:=CHR(ORD(CH1)-32)
  ELSE BIGLETTER:=CH1
END
```

Beachte, dass die Parameterdefinition in einer Funktion genau gleich wie in einer Prozedur erfolgt.

Kap. 3: Standard-Prozeduren

Eine Reihe von Prozeduren sind direkt im Compiler vordefiniert. Diese muessen nicht genau der Syntax von normalen Prozeduren folgen. Insbesondere koennen die Prozeduren READ und WRITE eine beliebige Anzahl Parameter haben.

3.1 Die Standardprozedur READ

Als erstes Argument kann ein File stehen:

```
READ(@KEY,CH1);
```

Dieses muss mit einem @-Zeichen gekennzeichnet werden.
Falls kein File steht, wird per Default vom Keyboard-
Buffer gelesen.

Weiterhin sind in beliebiger Reihenfolge und Zahl die
folgenden VARIABLEN PARAMETER, d.h. Namen von Variablen
moeglich:

Typ CHAR: Liest den naechsten Buchstaben
(kann auch z.B. BL oder CR sein).

Typ PACKED CHAR: Liest genau zwei Buchstaben.

Typ INTEGER: Zuerst werden, wenn noetig, Blanks
ueberlesen, dann wird eine Zahl
eingelesen. AM ENDE WIRD NOCH DER
ERSTE BUCHSTABE, DER NICHT MEHR
ZUR ZAHL GEHOERT, EINGELESEN.

Beispiel:

```
READ(CH1,NUM,TOKEN)  
mit folgendem Input: a -500abcdefa  
ergibt CH1='a', NUM=-500, TOKEN='bc'
```

Falls kein File angegeben ist, erfolgt die Eingabe
ueber einen Zeilenbuffer vom Keyboard (Mit Editier-
funktionen). Beachte, dass in diesem Fall jedes READ-
Statement eine neue Input-Zeile verlangt. Falls dies
nicht erwünscht ist, kann das Abfragen einer neuen
Zeile mit READ(@INPUT,...) unterdrückt werden.

3.2. Die Standardprozedur WRITE

Als erstes Argument kann wie bei READ ein File
stehen:

```
WRITE(@PRINTER,CR,LF);
```

Folgende Argumente (CONSTANTE PARAMETER, d.h beliebige
Ausdrücke) sind in beliebiger Zahl und Reihenfolge
möglich:

Typ CHAR: Schreibt einen Buchstaben aus (Auch Kontroll-
buchstaben).

Typ PACKED CHAR: Schreibt zwei Buchstaben aus.

Typ INTEGER: Schreibt eine Integer-Zahl.

Strings: 'xxxxxxxx' schreibt den String aus.

3.3 Die Standardprozedur CALL(ADDRESS: INTEGER)

Diese Prozedur erlaubt es, Maschinensprachprogramme an der Adresse ADDRESS als Subroutinen aufzurufen.

3.4 Die Standardprozedur OPENR(VAR DEVICE: FILE)

Mit dieser Prozedur kann ein TEXTFILE zum Lesen eroeffnet werden. Dazu muss jedoch zuerst FILENAME aufgeruft oder die entsprechenden Daten vorbereitet werden.

3.5 Die Standardprozedur OPENW(VAR DEVICE: FILE)

Eroeffnet ein TEXTFILE zum schreiben.

3.6 Die Standardprozedur FILENAME

Diese Prozedur schreibt ein = -Zeichen. Anschliessend kann ein Filename, ein Zyklus, ein Drivecode und eine Tape-Lokation eingegeben werden. Dient zum Vorbereiten der Prozeduren OPENR und OPENW.

3.7 Die Standardprozedur CLOSE(DEV: FILE;

Es koennen beliebig viele Textfiles abgeschlossen werden.

**3.8 Die Standardprozedur GETSEC(CONST SECTOR, TRACK,
ADDRESS: INTEGER; VAR ERRCODE: INTEGER)**

Diese Prozedur erlaubt das direkte Einlesen eines Disk-Sektors (Siehe Library DISKLIB).

**3.9 Die Standardprozedur PUTSEC(CONST SECTOR, TRACK,
ADDRESS: INTEGER; VAR ERRCODE: INTEGER)**

Erlaubt das direkte Schreiben eines Disksektors. (Siehe Library DISKLIB).

01 FILE READ ERROR
02 CHECKSUM ERROR
03 EXIT FROM READ/WRITE
04 RECORD NUMBER WRONG ON TAPE
05 FILE TYPE WRONG
06 FILE NOT FOUND
07 DISK NOT READY
08 DIRECTORY OVERFLOW
23 TOO MANY OPEN FILES (Max. 3)
24 DIRECTION ERROR OR WRITE PROTECT
25 FILE NOT OPEN
26 DISC OVERFLOW
27 TEXT FILE TOO LONG
31 DIVISION BY ZERO
32 STACK OVERFLOW
33 NOT A PASCAL PROGRAM
34 ILLEGAL P-CODE DETECTED

Kap. 5: LIBRARY

Das R65 LEVEL I PASCAL erlaubt die Erzeugung von Library-Files. Diese koennen Konstanten- und Variablendefinitionen, Prozeduren, Funktionen und eine Initialisierung enthalten. Libraries werden ebenfalls mit dem Compiler uebersetzt, der aber anstelle des normalen Loader-Files ein Library-Loader-File und ein Compiler-Library-File erzeugt. BEIDE FILES MUessen VORHANDEN SEIN, UM EIN PROGRAMM, DASS DIESE LIBRARY BENUETZT, ZU COMPILENIEREN UND ZU LADEN!

5.1 Library-Generierung:

Folgende Syntax wird als Compiler-Source verwendet:

```
LIBRARY library name;
CONST library constants;
VAR   library variables;
PROCEDURE any library Procedure;
.....
..... (any number of procedures and functions)
BEGIN
  (initialisation)
END.
```

Beachte, dass alle Konstanten, Variablen, Prozeduren und Funktionen, die auch einem mit der gleichen Syntax aufgebauten Programm zur Verfuesung staendenden, spaeter verwendet werden koennen.

5.2 Der Aufruf von Libraries:

Der Aufruf von Libraries muss unmittelbar nach dem Programmkopf erfolgen:

```
PROGRAM SYSTEMLIBRARYTEST;
USES SYSLIB;
CONST .....
```

Beachte, dass nur ein USES-Statement stehen darf,
dass aber mehrere Libraries aufrufen kann.

```
USES SYSLIB,DISKLIB;
```

5.3 Standardlibraries:

Die folgenden Libraries sind gegenwaertig vorhanden:

SYSLIB: Enthaeilt alle wichtigen Elemente zur Arbeit
mit dem R65 Computersystem.
PLOTLIB: Zur Arbeit mit dem Graphics-Display
DISKLIB: (In Arbeit: RANDOM ACCESS FILES etc.)

Kap. 6: R 6 5 P - C O D E S

00 STOP	01 RETN	02 NEGA	03 ADDA
04 SUBA	05 MULA	06 DIVA	07 LOWB
08 TEQU	09 TNEQ	0A TLES	0B TGRE
0C TGRT	0D TLEE	0E ORAC	0F ANDA
10 EORA	11 NOTA	12 LEFT	13 RIGH
14 INCA	15 DECA	16 COPY	17 PEEK
18 POKE	19 CALA	1A RLIN	1B GETC
1C GETN	1D PRTC	1E PRTN	1F PRTS
20 LITB	21 INCB	22 LITW	23 INCW
24 JUMP	25 JMPZ	26 JMPO	27 LOAD
28 LODX	29 STOR	2A STOX	2B CALL
2C SDEV	2D RDEV	2E FNAM	2F OPNR
30 OPNW	31 CLOS	32 PRTI	33 GHGH
34 GLOW	35 PHGH	36 PLOW	37 GSEC
38 PSEC			

Kap. 7: RESERVIERTE NAMEN

AND ARRAY BEGIN CALL CASE CHAR CHR CLOSE
CONST DIV DO DOWNT0 ELSE END FILE FILENAME
FOR FORWARD FUNC GETSEC HIGH IF INTEGER

```
LIBRARY LOW MEM NOT ODD OF OPENR OPENW  
OR ORD PACKED PROC PROGRAM PUTSEC READ  
REPEAT SHL SHR THEN TO UNTIL USES VAR  
WHILE WRITE XOR
```

Kap. 8: VORWAERTSREFERENZEN

Vorwaertsreferenzen sind in einem Fall noetig:
Nehmen wir an, dass eine Prozedur A eine Prozedur B aufruft, und umgekehrt. Dies laesst sich wie folgt verwirklichen:

```
PROC A;  
  PROC B;  
    BEGIN (B) ... A .... END (B);  
  BEGIN (A) .. B .. END (A);
```

Nun ist aber B nur noch in A gueltig, d.h. B kann nicht mehr von aussen aufgerufen werden. Falls dies jedoch noetig ist, muessen A und B gleichwertig definiert werden:

```
PROC A; FORWARD;  
PROC B;  
  BEGIN .... A .... END;  
PROC A;  
  BEGIN .... B .... END;
```

Beachte, dass eine Vorwaertsreferenz nur innerhalb eines Blockes moeglich ist. Ausserdem darf die Vorwartzreferenzierte Prozedur vorlaeufig KEINE PARAMETER haben, d.h.

PROC A(X); FORWARD ist illegal.

R65 P-CODES
=====

1 BYTE INSTRUCTIONS

00	STOP	STOP EXECUTION
01	RETN	RETURN FROM PROCEDURE
02	NEGA	NEGATE ACCU
03	ADDA	ADD ACCU TO STACKTOP
04	SUBA	SUBTRAKT ACCU FROM STACKTOP
05	MULA	MULTIPLY ACCU WITH STACKTOP
06	DIVA	DIVIDE STACKTOP BY ACCU
07	LOWB	MASK LOW BIT (INTEGER TO BOOLEAN)
08	TEQU	TEST EQUAL
09	TNEQ	TEST NOT EQUAL
0A	TLES	TEST LESS
0B	TGRE	TEST GREATER OR EQUAL
0C	TGRT	TEST GREATER
0D	TLEE	TEST LESS OR EQUAL
0E	ORAC	LOGICAL OR ACCU WITH STACKTOP
0F	ANDA	LOGICAL AND ACCU WITH STACKTOP
10	EORA	EXCLUSIVE OR ACCU WITH STACKTOP
11	NOTA	NOT ACCU
12	LEFT	LEFT SHIFT (ACCU TIMES)
13	RIGH	RIGHT SHIFT (ACCU TIMES)
14	INCA	INCREMENT ACCU BY ONE
15	DECA	DECREMENT ACCU BY ONE
16	COPY	COPY ACCU TO STACKTOP
17	PEEK	LOAD BYTE FROM (ACCU) "ABSOLUTE"
18	POKE	STORE BYTE TO (ACCU) "ABSOLUTE"
19	CALA	CALL ML ROUTINE AT (ACCU)
1A	RLIN	FILL LINE INPUT BUFFER WITH LINE
1B	GETC	GET CHAR
1C	GETN	GET DEZIMAL NUMBER
1D	PRTC	PRINT CHAR
1E	PRTN	PRINT DEZIMAL NUMBER
1F	PRTS	PRINT STRING

2 BYTE INSTRUCTIONS

20 X	LITB	LOAD LITERAL BYTE TO ACCU
21 X	INC B	ADD ONE BYTE (SIGNED!) TO SP

3 BYTE INSTRUCTIONS

22 XY	LITW	LOAD LITERAL WORD TO ACCU
23 XY	INCW	ADD WORD TO SP
24 XY	JUMP	JUMP TO LOCATION XY
25 XY	JMPZ	JUMP, IF LOW BIT ZERO
26 XY	JMPO	JUMP, IF LOW BIT ONE

4 BYTE INSTRUCTIONS

27 X YZ	LOAD	LOAD WORD
28 X YZ	LODX	LOAD INDEXED WORD
29 X YZ	STOR	STORE WORD
2A X YZ	STOX	STORE INDEXED WORD
2B X YZ	CALL	CALL PROCEDURE