Kurzbeschreibung PICO-PASCAL

Version 1.17

Klaus Rumrich
Spessartstr. 3
63477 Maintal
Tel. 06109/62868

PICO-PASCAL ist eine Untermenge von PASCAL, die bereits die wichtigsten Sprachelemente von PASCAL enthält. Daß es sich auch für größere Programmprojekte eignet, beweist die Tatsache, daß der vorliegende Compiler vollständig in PICO-PASCAL geschrieben wurde (über 5000 Zeilen PASCAL-Quelltext).
Der Compiler erzeugt Assembler-Quelltext, der anschließend vom Grundprogrammassembler übersetzt werden kann.
Das Ergebnis ist ein File im JADOS-Objektformat, das unter Verwendung des JADOS-Linkers mit der Laufzeitbibliothek zum lauffähigen Programm gelinkt wird.

Sowohl der Compiler als auch alle übersetzten Programme sind auf allen Prozessoren 680xx unter JADOS lauffähig und vollständig relokativ.
Als Option kann eingestellt werden, daß das Programm auf einem 68020 laufen soll; es werden dann auch spezielle 68020-Befehle benutzt. Solche Programme sind dann nur noch auf diesem Prozessor lauffähig.

Der Compiler kennt derzeit folgende Typen:
INTEGER, BOOLEAN, CHAR, STRING, TEXT, Aufzählungstypen, Unterbereichstypen, beliebigdimensionale ARRAYs, Pointer und RECORDs.
SETs und FILEs fehlen noch. Derzeit gibt es als einzigen Dateityp den Typ TEXT.
Der Typ REAL läuft sowohl mit als auch ohne FPU. Gleitkommazahlen werden in beiden Fällen im gleichen Format abgelegt, allerdings werden ohne FPU nur 32 Bit Mantisse benutzt (mit FPU: 53 Bit Mantisse).

Reservierte Worte

AND	ARRAY	ASM	BEGIN
CASE	CONST	DIV	DO
DOWNTO	ELSE	END	EXTERNAL
FILE *	FOR	FORWARD	FUNCTION
GLOBAL *	GOTO *	IF	IN *
LABEL *	MOD	NOT	OF
OR	OTHERWISE	PACKED	PROCEDURE
PROGRAM	RECORD	REPEAT	SET *
SHL	SHR	THEN	TO
TYPE	UNTIL	VAR	WHILE
WITH	XOR		

Die mit einem Stern (*) versehenen Worte sind bereits reserviert, werden aber noch nicht benutzt. Sie sind für spätere Erweiterungen vorgesehen.
Das Schlüsselwort PACKED ist nur aus Kompatibilitätsgründen vorhanden, da grundsätzlich alle ARRAYs of CHAR gepackt sind und alle anderen ARRAYs nicht.
Die nicht explizit aufgeführten Fälle in CASE bekommt man sowohl mit ELSE als auch mit OTHERWISE.
Mit ASM(Assemblerbefehl) kann ein Assemblerbefehl direkt in das Zielfile geschrieben werden. Es wird bis zur ersten schließenden Klammer ausgegeben, deshalb sind Assemblerbefehle, die Klammern enthalten, nicht möglich.

Vordefinierte Symbole

Konstanten:	TRUE	FALSE
	MAXINT	
	NIL	
Typen:	INTEGER	(2 Byte)
	BOOLEAN	(1 Byte)

```
            CHAR        ( 1 Byte )
            POINTER     ( 4 Byte )
            REAL        ( 8 Byte )
            STRING      ( 82 Byte )
            TEXT

Variablen:  INPUT       OUTPUT

Funktionen: ORD         CHR
            SUCC        PRED
            ABS
            SQR
            ODD
            EOF         EOLN
            KEYPRESSED  GETKEY
            LENGTH

            INT         TRUNC       ROUND
            SQRT
            EXP         LN +
            SIN +       COS +       TAN +
            SINH +      COSH +      TANH +
            ARCSIN +    ARCCOS +    ARCTAN +

            + = Nur mit FPU verwendbar

Prozeduren: WRITE       WRITELN
            READ        READLN
            RESET       REWRITE     CLOSE
            HALT
            INC         DEC
            NEW         DISPOSE
```

Bemerkungen:

Der Vorrang der Operatoren:

1. Unäre Operatoren
   Vorzeichen (+ und -) NOT
2. Multiplikationsartige Operatoren
   * / DIV MOD AND SHL SHR
3. Additionsartige Operatoren
   + - OR XOR
4. Vergleichsoperatoren
   = <> < > <= >=

Innerhalb jeder Gruppe wird die Berechnung von links nach rechts ausgeführt (linksassoziativ).
Durch explizite Klammerung kann von obiger Reihenfolge abgewichen werden.
Die Reihenfolge der Berechnung von Teilausdrücken ist nicht festgelegt und kann sich in späteren Compiler-Versionen ändern. Programme, die von Berechnungsreihenfolgen innerhalb eines Ausdruckes abhängen, sind zu vermeiden.

Offene Files werden am Programmende NICHT automatisch geschlossen.
Der Programmierer hat explizit ein CLOSE(datei) anzugeben, wenn auf eine Datei geschrieben wurde.

RESET und REWRITE müssen stets in der Form RESET(datei,dateiname) benutzt werden. Die Form RESET(datei) wird zwar vom Compiler akzeptiert, ist jedoch noch nicht in PASLIB.ASM vorhanden.

DISPOSE hat keine Wirkung. Mit NEW angeforderter Speicherplatz auf dem Heap kann daher NICHT mehr freigegeben werden.

READ und READLN von der Tastatur arbeiten mit einem Eingabepuffer.
Erst nach Betätigen der Return-Taste wird die gesamte Zeile an das Programm weitergegeben.
Es wird dazu die JADOS-Funktion LINEEDIT benutzt.

INTEGER-Konstanten können auch hexadezimal mit vorangestelltem "$" angegeben werden.
CHAR-Konstanten können auch durch Angabe des ASCII-Codes mit vorangestelltem "#" angegeben werden.

Stringvergleiche sind noch nicht möglich (wohl aber Vergleiche
von ARRAY OF CHAR).

Pointerwertige Funktionen sind noch nicht erlaubt.

Argumente von Funktionen und Prozeduren dürfen in lokalen
Unterprogrammen nicht global übergeben werden. Der Compiler
erzeugt dabei falsche Adressen.

Variante Records sind nicht erlaubt.
In Record-Deklarationen muß die letzte Komponente vor dem
END mit einem Semikolon abgeschlossen werden.

Abweichend von Standard-Pascal können Deklarationen mit
CONST, TYPE, VAR, FUNCTION, PROCEDURE in beliebiger
Reihenfolge und auch mehrmals vorkommen. Dies vereinfacht
die Benutzung von Include-Dateien.

Abweichend von Standard-Pascal können die Operatoren
NOT, AND, OR auch auf INTEGER-Operanden angewandt werden.
Die Operation wird dann Bitweise durchgeführt.

Abweichend von Standard-Pascal sind auch die Operationen
SHL (bitweises links-schieben), SHR (bitweises rechts-schieben)
und XOR (bitweise Exclusiv-Oder Verknüpfung) mit INTEGER-
Operanden möglich.

Bei der Ein- und Ausgabe von REAL-Werten darf die REAL-Variable
bzw. der REAL-Ausdruck nicht das erste Argument der Funktionen
READ/READLN/WRITE/WRITELN sein. Gegebenenfalls muß als erstes
Argument die Dateivariable INPUT bzw. OUTPUT angegeben werden.
Beispiel: (X ist REAL)
Anstatt READLN(X) muß READLN(INPUT, X)
benutzt werden.
Anderenfalls wird das Programm ohne Fehlermeldung compiliert,
stürzt aber bei Ausführung der entsprechenden Anweisung ab.

Die Ausgabe von REAL-Werten mit WRITE/WRITELN ist derzeit nur
mit FPU möglich.
Die Eingabe von REAL-Werten mit READ/READLN ist auch ohne FPU
möglich (hoffe ich jedenfalls).

Weitere Prozeduren und Funktionen in PASLIB.LIB:
(müssen vor Benutzung mit EXTERNAL deklariert werden)

Grafik:	MOVETO	DRAWTO		
	NEWPAGE	SETFLIP	AUTOFLIP	
	CLPG	CLR		
	ERAPEN	SETPEN		
	RECTANGLE	BAR	LINE	CIRCLE
	PROGZGE			
	SETXOR			

Schildkrötengrafik:			
	SCHREITE	SCHR16TEL	DREHE
	HEBE	SENKE	
	FIRSTTIME	GRAPOFF	
	GRAPHIDE	GRAPSHOW	
	GRAPSET	AUFXY	AUFK

Text:	G_WRITE	G_READ	(Grundprogramm-WRITE und -READ)
	READAUS		
	G_VERSION		(Grundprogrammversion ermitteln)
	CLRSCREEN		
	SETCURXY	GETCURXY	
	CURON	CUROFF	CURSEIN CURSAUS
	CLREOLN		
	GETLINE		
	SIZE		

```
Syboltabelle:
            SYMCLR      WERT       FPUWERT   ZUWEIS

Strings:    STRUPPER
            STRAPPEND   STRDELETE
            STRPOS      SUBSTRING

Mathematik: IPOWER

Uhren:      GETTIME     GETDATE    UHRPRINT

Geräte:     GETAD8      SETDA
            REL         SERINIT
            GDPVERS

JADOS:      J_VERSION             (Jados-Version bestimmen)
            BEEP        BELL       ERRNOISE   SOUND
            WRITECMD
            GETPARM
            SETRETCODE  GETRETCODE
            CPU
            MOTOROFF
            RESPONSE
            INPORT      OUTPORT
            FLOPPYRD    FLOPPYWR
            DRIVECODE   GETDRIVE   SETDRIVE
            GETDPATH
            GETHDISK    DISKINFO   FILEINFO
            CHMODE      ERASE      RENAME
            APPEND               (Datei zum Anhängen öffnen, funktioniert
                                  nur mit Dateien, die wirklich im letzten
                                  gespeicherten Sektor enden!!)

Sonstiges:  RANDOM
            DELAY       SYNC
            CALLDUMP

Direktaufruf Grundprogramm und JADOS:
            GRUND       GRUND1     JADOS
```

Die Deklaration dieser Funktionen und Prozeduren kann mit dem Include-File GRUND.INC vorgenommen werden. Dort kann auch die genaue Syntax für den Aufruf und der Typ der Argumente nachgesehen werden.

Gerätenamen

Mit RESET(textvariable,dateiname) oder REWRITE(textvariable,dateiname) wird eine Textvariable einer Datei zugeordnet.
Einige Dateinamen bezeichnen Ein-/Ausgabegeräte:

	In	Out	
CON:	X		Console bzw. Tastatur
CRT:		X	Bildschirm
LST:		X	Drucker
SER:	X	X	Serielle Schnittstelle
NIL:		X	Dummy
CAS:	X	X	Kassettenschnittstelle (in späteren Versionen)
RAM:	X	X	RAM-Floppy (")

Compiler-Optionen

Compiler-Optionen werden immer im PASCAL-Quelltext eingestellt.

Eine Compiler-Option wird in Kommentarklammern eingeschlossen,
das erste Zeichen nach der öffnenden Kommentarklammer ist ein
Dollarzeichen ("$").

Das nächste Zeichen bestimmt die Option, ein folgendes "+" oder
"-" Zeichen schaltet die Option gewöhnlich ein oder aus.

Vorhandene Optionen:

(*$C+ *) (*$C- *) CPU 68020
Es wird Code für den 68020 erzeugt.
Das Programm ist nicht lauffähig auf
68008 und 68000.

(*$D+ *) (*$D- *) Debug
Diese Option ist nur am Programmanfang
erlaubt.
Bei eventuellen Programmabstürzen kann
kontrolliert werden, welche Prozeduren und
Funktionen aufgerufen wurden.

(*$F+ *) (*$F- *) Far Calls
Es werden weite Sprünge mit 32-Bit Offset
erzeugt (auch möglich bei 68008 + 68000).
Nötig bei großen Programmen.

(*$I dateiname *) Include
Der Inhalt der Datei "dateiname" wird
an dieser Stelle eingefügt. Die restliche
Zeile nach der Option wird ignoriert.

(*$K+ *) (*$K- *) Kommentare
Der PASCAL-Quelltext wird als Kommentar
in das Assembler-Ausgabefile geschrieben.

(*$L+ *) (*$L- *) MOVE.L erzegen
Es werden MOVE.L Befehle zum Kopieren von
Arrays benutzt. Nur mit 68020 benutzen !!!

(*$M stacksize,heapsize *) Speicherbereiche
Diese Option ist nur am Programmanfang
erlaubt.
stacksize und heapsize werden in KByte
angegeben und definieren die Größe des
Stacks und des Heaps. Zur Laufzeit wird
überwacht, daß der Stack nicht vom Heap
zerstört wird.
šberlauf des Stacks wird allerdings nicht
kontrolliert.
Voreinstellung: Stack 16 KByte
Heap 16 KByte

(*$N+ *) (*$N- *) FPU verwenden
Zur Zeit nicht benutzt.

(*$R+ *) (*$R- *) Rangecheck
Bei Indizierung von ARRAYs wird geprüft,
ob der Index innerhalb des zulässigen
Bereichs liegt.

(*$Z+ *) (*$Z- *) Warnings anzeigen
Der Compiler zeigt Warnings an, z.B.:
unbenutzte Typen/Variablen/Funktionen
Eine genauere Auflistung wird in die
optionale Logdatei geschrieben.

Aufruf des Compilers

Der Compiler kann entweder durch

PASCAL

oder durch

```
  PASCAL quellfile zielfile [logfile]
```

gestartet werden. In der Form ohne Parameter wird das File erfragt
(Eingabe ohne .PAS). In der Form mit Parameter müssen beide Files
angegeben werden (mit Extension, gewöhnlich .PAS und .ASM).
Die Angabe einer Logdatei ist optional.
Der Compiler benutzt den JADOS-Returncode, um dem System mitzuteilen,
ob Fehler aufgetreten sind. Mit Hilfe des Programmes RETCOD.COM
kann dieser Returncode abgefragt werden oder eine bedingte
Verarbeitung aufgerufen werden, siehe Beispiel in P.BAT.
Das vom Compiler erzeugte Zielfile muß vom Grundprogrammassembler
übersetzt werden und anschließend mit der Laufzeitbibliothek
PASLIB.LIB gelinkt werden.
Am einfachsten geht der gesamte Vorgang mit der Batchdatei P.BAT.
Der Aufruf erfolgt durch

```
  P datei
```

und startet den Compiler, der Assemblertext wird auf das File
$PASOUT.ASM geschrieben, anschließend assembliert und mit der
Laufzeitbibliothek PASLIB.LIB gelinkt.
Das ausführbare Programm heißt anschließend datei.68K.

Soll die Batchdatei die Ramfloppy oder Festplatte als Zwischen-
speicher benutzen, so ist die Datei entsprechend zu ändern.

Der Compiler befindet sich zwei mal auf der Diskette.
PASCAL.68K läuft auf allen Prozessoren 680xx, während
PASCAL20.68K nur auf 68020 lauffähig ist.
Der Compiler wurde mit eingeschalteter Rangecheck-Option
übersetzt. Falls während einer Compilierung eine
CHK-Exception auftritt, ist irgendein ARRAY im Compiler
falsch indiziert worden. Dies sollte im Normalfall eigentlich
nicht vorkommen, da die Arraygrößen vom Programm überwacht
werden. Allerdings tritt nach einigen Typfehlern (insbesondere
bei Gleitkommaarithmetik) eine derartige Fehlermeldung
auf, wenn die šbersetzung fortgesetzt wird.

Die Bibliothek PASLIB.LIB

Die Laufzeitbibliothek ist im Quelltext vorhanden, kann aber nicht
mit dem normalen Grundprogrammassembler übersetzt werden.
Nach ähnlichem Muster kann aber eine eigene Bibliothek mit
Assembler-Routinen erstellt werden. Die entsprechenden Routinen
brauchen dem Compiler lediglich mit EXTERNAL-Deklarationen
bekannt gemacht werden und die neue Bibliothek dem Linker angegeben
werden.

Achtung: Assembler-Unterprogramme dürfen keine Register verändern!

Ein weiteres, besonders einfaches und kommentiertes Beispiel
eigener Assembler-Unterprogramme befindet sich in OWN.ASM in
Verbindung mit OWNTEST.PAS.

Änderungen zu Version 1.14

Bei eingeschalteter Debug-Option werden nach
Laufzeitfehlern auch Zeilennummern angezeigt
(Nach Auswahl von "C=CALLDUMP").
REAL-Arithmetik auch ohne FPU
REAL-Konstanten in Ausdrücken
Erweiterte Laufzeitbibliothek

Änderungen zu Version 1.12

Rückgabe eines Return-Code an das System
NOT, AND, OR auch mit INTEGER-Operanden
Neue reservierte Worte SHR, SHL, XOR

Verwendung einer Logdatei
READ/READLN von Tastatur mit Eingabepuffer für eine Zeile

Änderungen zu Version 1.5

Nur für benötigte Funktionen wird EXTPROC erzeugt
JADOS-Objektformat wird unterstützt
WITH implementiert
IF FALSE, WHILE TRUE etc. korrigiert
Neue Optionen: D,L,Z
Anzeige während des Compilerlaufes verbessert
Fehlerbehandlung verbessert

Änderungen zu Version 1.2

Gleitkommaarithmetik lauffähig mit FPU
FOR..TO..DO jetzt auch mit CHAR oder BOOLEAN
Berechnung konstanter Ausdrücke zur Compile-Zeit
Effizientere Adressierung und Datenzugriff
Erzeugter Code 20% kürzer

Vorgesehene Erweiterungen/Verbesserungen

READ/READLN mit ARRAY OF CHAR
Gleitkommakonstanten in CONST-Anweisungen
WRITE/WRITELN mit REAL ohne FPU
Funktionierendes DISPOSE
Variante Records (?)
SHORTINT (1 Byte) und LONGINT (4 Byte) (?)

Gleitkommaarithmetik lauffähig mit FPU
FOR..TO..DO jetzt auch mit CHAR oder BOOLEAN
Berechnung konstanter Ausdrücke zur Compile-Zeit
Effizientere Adressierung und Datenzugriff
Erzeugter Code 20% kürzer