Informationstechnik für Ingenieure

PASCAL

Grundlegendes:

- PASCAL beruht auf Englisch (Groß-/Kleinschreibung ist relevant!!)
- Zeichenvorrat: 1)gesamtes Alphabet, Ziffern 0-9

2) Sonderzeichen: + , - , * , / , < , > , = , . , : , ; , ( ) , [ ] , { } , ^

dabei wichtig:

.	Ende des Programms
;	Ende einer Befehlseinheit
:=	Wertzuweisung einer Variable
( )	für Parameterbestimmung
[ ]	für Array-Indizes
{ }	klammert Programmteil vom Kompilieren aus

3) logische Operatoren (zum Verknüpfen):

and	logisches ´und´
or	logisches ´oder´
not	logisches ´nein´

4) arithmetische Operatoren (Rechenverfahren):

+	Addition, Vorzeichen
-	Subtraktion, Vorzeichen
*	Multiplikation
/	Division
div	ganzzahlige Division
mod	ganze Restbildung bei ganzzahliger Division

5) Vergleichsoperatoren:

=	Gleichheit
<>	Ungleichheit
>	größer
<	kleiner
>=	größer gleich
<=	kleiner gleich

6) weiteres Vokabular:

begin	:	Anfang einer größeren Befehlsstruktur
end	:	Ende einer größeren Befehlsstruktur
file	:	Datei (Daten o.ä.)

Programmaufbau:

- Programmkopf [program <name>(Ein-/Ausgangsparameter-Parameter]
- Deklarationsteil
  - Konstanten [const x = ...]
  - Typen [type x = ...]
  - Variablen [var x : ...]
  - Prozeduren & Funktionen [procedure <name>(Parameter)] & [function <name>(Parameter)]
- Hauptprogramm [begin ->Aufbau (incl. Prozeduren & Funktionen) end.]

Vorhandene Daten- Eingabetypen:

Die verwendeten Variablen werden als einem Datentyp zugehörig deklariert: diese müssen deklariert werden (siehe unten), falls nicht einer der folgenden Standard – Datentypen genügt:

- integer : aus dem Bereich der ganzen Zahlen
- real : aus den reellen Zahlen
- boolean : kann 2 Zustände annehmen (z.B. true / false)
- char: kann als Wert ein Zeichen annehmen (aus der ASCII – Tabelle)
- string : Buchstaben-/Zeichenkette (später)

- Die Ausgabe (auf dem Monitor) erfolgt durch: write(output,‘.....‘); **[**´writeln´ erzeugt einen Zeilenumbruch**]**
- Die Einabe (Tastatur)durch: read(input,...); **[**´readln´ erfordert RETURN nach der Eingabe**]**

Diese Prozeduren gelten nur für sie Standard-Datentypen!!
Anzahl der Spalten / Nachkommastellen: Doppelpunkt hinter der auszugebenden Variable, dann gewünschte Anzahl eingeben

- Daten können auch aus Dateien eingelesen (bzw. darauf geschrieben)werden:

Einlesen:		Schreiben:	
	reset(Parameter,‘<dateiname>‘);		rewrite(Parameter,<dateiname>);
	read(Parameter, Variable);		write(Parameter,‘...‘);
	close(Parameter);		close(Parameter);

Schleifen:

Mittels Schleifen (Kontrollstrukturen) können verschiedene Algorithmen zur Problemlösung realisiert werden.
Das Grundkonzept ist dabei die Wiederholung von Befehlsfolgen, bis eine Bedingung erfüllt ist.
Es existieren folgende Schleifen – Typen:

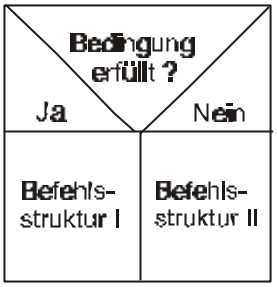
- **if-Schleife:**

Anwendung

Syntax

```
if (Bedingung) then
  begin
    Befehlsstruktur I ;
  end
else
  begin
    Befehlsstruktur II ;
  end;
```

Struktogramm



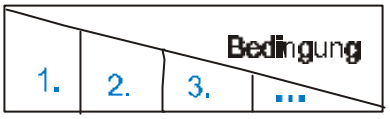
- **case-Schleife:**

Anwendung

Syntax

```
case (Bedingung) of
   1 : Befehl;
   2 : Befehl;
   :       ;
end;
```

Struktogramm



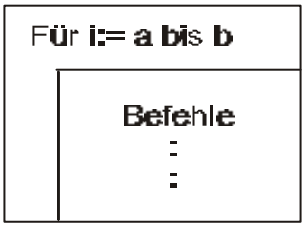
- **for-Schleife:**

Anwendung

Syntax

```
for (index:= a) to(b) do
begin
    Befehle;
end;
```

Struktogramm



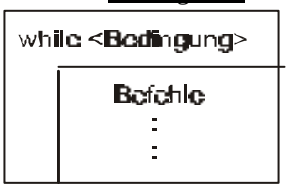
- **while-Schleife:**

Anwendung

Syntax

```
while (Bedingung) do
begin
  Befehle;
end;
```

Struktogramm



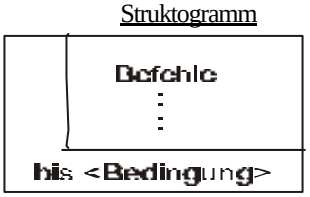
- **repeat-Schleife:**

Anwendung

Syntax

```
repeat
    Befehle;
until (Bedingungen);
```

Struktogramm



Prozeduren und Funktionen

Prozeduren sind eigenständige Unterprogramme, die Teile eines Programms abarbeiten und die benötigten Daten wieder an das Hauptprogramm zurückgeben. Sie sind auch in mehreren Programmen einsetzbar (Softwarerecycling).
Funktionen geben einen Wert an das Hauptprogramm / bzw. Programmteil zurück ; verwendet bei häufig benötigten Berechnungen (z.B. Eingabevorgänge, Wertberechnungen, etc.).

- *Aufbau Prozedur:* procedure <name>(globale Parameterliste);

  wichtig!!
  1) Variablendeklaration: alle Parameter müssen einem Typ zugeordnet sein
  2) Globale Variablen (im Hauptprogramm bekannt & deklariert)
     a) call by reference: Variable wird in Prozedur geändert (Dekl.: var ...)
     b) call by value: Variable wird nicht geändert (Deklaration: <Variable>)
  3) lokale Parameter (nur in Prozedur bekannt) ; im lokalen Deklarationsteil
- *Aufbau Funktion:* function <name>(Parameterliste) <type>;

  - <name> entspricht der zu übergebenden Variablen
  - die Parameter sind alle call by value

Diese Unterprogramme können aufgerufen werden – entweder im Hauptprogramm oder untereinander (nur schon bekannte Prozeduren dürfen eingebunden werden).
Der Aufruf erfolgt so: **<name>(Aktualparamterliste)**; dabei müssen die Aktualparameter des gleichen Typs, in der gleichen Reihenfolge und im Hauptprogramm sowie der Prozedur als globale Parameter bekannt sein.

Datentypen

Jede Variable muß einem Typen zugeordnet sein (Standardtypen: integer, real, char, boolean). Die Zuordnung steht im Deklarationsteil ´type´ mit : **<name> = <type>** (auch Unterbereichstypen möglich). Wenn <type> eine Menge ist, steht diese in Klammern ( ); sie wird automatisch numerisch geordnet . Mit diesen Ordungszahlen und den Befehlen **pred( )** & **succ( )** & **ord( )** kann gearbeitet werden.
Weitere Operationen sind mit den *Relationsoperatoren* möglich:

=	Test auf Mengengleichheit
<>	Test auf Mengenungleichheit
<= , >=	Test auf teilweise Mengengleichheit
in	Test auf Beinhaltung eines Elements
set of	Typ ist eine Zusammenstellung der Elemente einer zuvor definierten Menge

höhere Datentypen:

- *array:*
  - Vektor eines Typs
  - Deklaration: <name> = array[a..b] of <anderer Typ>
  - Aufruf als: Parameter[Stelle] ; mit for-Schleife einlesen
- *matrix:*
  - Matrix (mehrdimensionaler Vektor) eines Typs
  - Deklaration: <name> = array[a..b,c..d] of <anderer Typ>
  - Aufruf als: Parameter[Zeile,Spalte] ; mit Doppel-for-Schleife einlesen
- *string:*
  - Zeichenketten vom Typ char ; durch ´ ` eingeschlossen
  - als Typ mit festgelegter Länge möglich: packed array[1..Länge] of char;
- *record*
  - Verbund mehrerer Typen
  - Deklaration: <name> = record
    <Unterbereiche> : <andere Typen> ;
    :
    end;
  - Aufruf: Parameter.Unterbereiche
  - Ausgabe von nicht-Basistypen muß mit einer case-Schleife erfolgen !!

Dateien (als Speicherplatz)

Im Programm können große Datenmengen in Dateien (zwischen)gespeichert werden: Daten können eingelesen und gespeichert , außerdem die entstandenen Dateien zur Weiterverarbeitung genutzt werden.

Dabei existieren 2 Arten von Dateien:

- *externe Datei:*
  - Textinhalt, der auch im Editor oder unter Unix lesbar ist
  - Deklaration: 1) Programmkopf: program <name>(input,output,datei)
    2) Variablendeklaration: datei : text ;
  - Bearbeitung:
    1) Daten einlesen:
       reset(datei,´<dateiname.dat>´) ;
       read(datei, <Variable>) ;
       close(datei) ;
    2) Datenspeichern:
       rewrite(datei,´<dateiname.dat>´) ;
       write(datei,<Variable>) ;
       close(datei) ;
- *interne Datei:*
  - programminterne Datenablage ; nicht lesbar
  - Deklaration: Typendeklaration: <name> = file of <anderer Typ> ;
  - Aufruf : siehe oben , aber auch mit get(<name>) und put(<name>) möglich
  - WICHTIG: Datei muß bei Programmstart erstellt (leer) sein !!

Befehle zum Bearbeiten:

eoln	=	Ende der Zeile (gibt logisches Ausdrücke zurück) ; zum Einlesen eines Arrays
eof	=	Ende der Datei (gibt logische Ausdrücke zurück) ; zur Beendigung des Einlesevorgangs

GNUPLOT

Bei der digitalen Grafikbearbeitung existieren zwei unterschiedliche Konzeptionen: *vektororientiert* (grafische Gestaltung mit mathematischen Formeln) und *pixelorientiert* (Definition von Farbpaketen).
Standards: vektororientiert: HPGL , DXF pixelorientiert: TIFF , PCX Drucker: postscript

Gnuplot – Dateien werden im Editor erstellt:

- *Programmaufbau:* - Systemgestaltung (Koordinatensystem, Beschriftung, Einstellungen) -> Befehle
  - plot ´<datei>´
  - Ausgabe:
    1) set terminal hpgl
       output ´pfad/<datei.hp>´
       replot
    2) set terminal ps
       output ´pfad/<datei.ps>
       ´replot
    3) set terminal x11

- *Syntax:*
  - wichtiges: # Zeile nicht Gnuplot-relevant
    / Fortführung einer Befehlszeile im Editor
  - besondere Rechenoperaten: ** Potenzierung
    = = Gleichheit
    ! = Ungleichheit
  - bekannte Funktionen: alle trigonometrischen Grundfunktionen
    log(x) , exp(x) , sqrt(x)
  - plot => 2-dimensional ; splot => dreidimensional
  - plot ´<datei.dat>´ using 3:4 => bildet 4. Spalte der Datei über 3. ab (sonst 1:2)

- *Befehle:*

- set xrange [a:b]	legt Intervall[a,b] für das Plotten fest
- set [no]x/ytics [a,b,c]	malt Striche im Abstand b für Intervall[a,c]
- set [no]autoscale	automatische Achsenskalierung
- set x/ylabel ´Achse´	schreibt ´Achse´ an die entsprechende Achse
- set title ´... ´	setzt den Titel
- set [no]polar	schaltet auf Polarkoordinaten um
- set trange, set rrange	legt Intervall für t und r bei Polarkoordinaten fest
- set samples 200	berechnet 200 Werte beim Plotten (sonst 100)
- set isosamples 5	plottet 5 Funktionslinien jeder Richung (sonst 10)
- set view 70,40	ändert den Blickwinkel in Grad (sonst 60,30)
- set hidden3d	blendet die verdeckten Lienen aus
- clear	entfernt alles vom Monitor
- set parametric	Umstellung auf parameterabhängige Darstellung

Darstellung komplexer Zahlen : x + y{0,1} => x +yi

Aufruf von GNUPLOT: gnuplot
Laden der zu bearbeitenden Datei: load ´pfad/<datei>´

Die entstandenen Dateien können wie folgt weiterverarbeitet werden:
<datei.hp> mit: lpr –Ppl1 <datei.hp> plotten
<datei.ps> mit: lpr –Ppss <datei.ps> drucken