Programmieren mit

PASCAL

Eine Kurzanleitung

Informatik Sekundarstufe II
Städtische Gesamtschule Duisburg-Hamborn/Neumühl
August 2000

1 EINLEITUNG 3

2 GRUNDLAGEN 3

2.1 AUFBAU EINES EINFACHEN PASCAL-PROGRAMMS 3
2.2 AUFBAU EINES KOMPLEXEN PASCAL-PROGRAMMS 4
2.3 VARIABLEN UND DATENTYPEN 5
2.4 VARIABLENDEKLARATION 5

3 ANWEISUNGEN UND KONTROLLSTRUKTUREN 5

3.1 EINGABEANWEISUNGEN 5
3.2 AUSGABEANWEISUNGEN 5
3.3 DIE ZUWEISUNG 5
3.4 DIE BEDINGTE ANWEISUNG (IF...THEN...ELSE) 6
3.4.1 Geschachtelte Fallunterscheidungen 6
3.4.2 Die Case-Anweisung 6
3.5 WIEDERHOLUNGEN (SCHLEIFEN) 6
3.5.1 Die for-Schleife 6
3.5.2 Die repeat-Schleife 7
3.5.3 Die while-Schleife 7

4 SELBSTDEFINIERBARE DATENTYPEN 7

4.1 DER AUFZÄHLUNGSTYP 7
4.2 DER AUSSCHNITTSTYP 7
4.3 DER SET-TYP 7
4.4 DER ARRAY-TYP (DATENTYP FELD) 8
4.5 DER RECORD-TYP (VERBUNDDATENTYP) 8

5 FUNKTIONEN UND PROZEDUREN 8

5.1 FUNKTIONEN 8
5.2 REKURSIVE FUNKTIONEN 9
5.3 PROZEDUREN 9
5.4 REKURSIVE PROZEDUREN 9

Vorbemerkungen

Diese Kurzanleitung führt in die wesentlichen Sprachkonzepte und Kontrollstrukturen der Programmiersprache Pascal ein. Die Kursanleitung ist als Nachschlagewerk bei der Programmierung von Pascalprogrammen im Informatikunterricht und bei Hausaufgaben gedacht. Anstatt die typischen Konzepte, Kontrollstrukturen und Merkmale der Programmiersprache systematisch einzuführen, wird häufig mit einfachen, eingängigen Programmbeispielen gearbeitet, die einen schnellen Einblick erlauben.

1 Einleitung

Die Programmiersprache Pascal wurde an der Eidgenössischen Technischen Hochschule Zürich unter der Leistung von Nikolaus Wirth um das Jahr 1970 entwickelt. Eine Programmiersprache ist eine Sprache, in der man einem Computer einen Algorithmus mitteilt.
Ein Algorithmus ist ein Verfahren zur systematischen, schrittweisen Lösung eines Problems.

2 Grundlagen

In diesem Abschnitt sollen zunächst einmal die Grundlagen des Aufbaus von Pascalprogrammen angegeben werden. Dabei wird zuerst der Aufbau eines sehr einfachen Programms betrachtet und im folgenden der Aufbau eines Aufwendigen, komplexen Pascal-Programms.

2.1 Aufbau eines einfachen Pascal-Programms

Ein Pascal-Programm besteht im Wesentlichen aus drei Teilen, einem Programmkopf, einem Deklarationsteil und einem Anweisungsteil.
Im Programmkopf wird der Programmname zur Identifizierung des Programms eingetragen. Dieser Programmname kann frei gewählt werden und muß nur den Anforderungen für die Schreibweise von Bezeichnern entsprechen.
Im Deklarationsteil werden die Variablen vereinbart, indem Variablenname (Bezeichnername) und datentyp festgelegt werden.
Im Anweisungsteil des Pascalprogramms stehen die eigentlichen Anweisungen des Algorithmus als Pascal-Befehle.
Kommentare erleichtern das Verständnis des Pascal-Programms und werden in geschwungenen Klammern notiert. Es wird empfohlen, wichtige Programmteile wie z.B. Variablendeklarationen, Anfang (begin) und Ende (end;) von Blöcken, Prozeduren, komplizierte Anweisungen und Berechnungen und „Programmiertricks“ zu kommentieren.
Ein Beispiel soll dies verdeutlichen.

```
program KleinesEinMalEins;

uses wincrt;        {Unit für Windows-Fenster}
var i : integer;    {Zählvariable bzw. Muliplikand}

    j : integer;    {Zählvariable bzw. Multiplikator}


begin {Hauptprogramm}

for i:=1 to 10 do

    begin

    for j:=1 to 10 do

        begin

        writeln(i,' * ',j,' = ',i*j)
        end; {for j}
    end;{for i}
end. {Hauptprogramm}
```

2.2 Aufbau eines komplexen Pascal-Programms

Das folgende Beispiel zeigt den Aufbau eines komplexen Pascal-Programms zur Verwaltung einer Siegerliste bei einem Ballonwettbewerb. Das Beispielprogramm demonstriert die folgenden Konstrukte:

- Deklaration von Konstanten mit dem Schüsselwort const
- Deklaration von Datentypen mit dem Schlüsselwort type
- Deklaration von Variablen mit dem Schlüsselwort var
- Deklaration von Funktionen mit dem Schlüsselwort function. (Prozeduren werden ähnlich deklariert.)
- Die Verwendung der Datenstruktur Array
- Die Kontrollstruktur „while ... do“

```
program ballonwettbewerb;
uses wincrt;          {Unit für Windows-Fenster Ein-/Ausgabe verwenden}
const                          {Deklaration der Konstanten}
      WortLaenge    = 10;
      MaxTeilnehmer = 1000;

type                           {Deklaration von selbstdefinierten Datentypen}
      TWort        = String[WortLaenge];          {Stringdatentyp der Länge 10}
      TTeilnehmer  = record                       {Verbunddatentyp}
                       Entfernung : integer;
                       Name       : TWort;
                       Vorname    : TWort;
                     end;
      TIndex       = 1..MaxTeilnehmer;            {Auszählungstyp}
      TSiegerliste = array[TIndex] of TTeilnehmer; {Kombinierter Datentyp}

var   EinTeilnehmer : TTeilnehmer;        {Variable m vom Typ TTeilnehmer}
      sliste        : TSiegerliste;       {Variable Sliste vom Typ TSiegerliste}
      s             : TIndex;             {Variable s vom Typ TIndex, siegerzahl}
      j             : TIndex;             {Zählvariable}

function liesTeilnehmerDatenOderEnde(Var Teilnehmer:TTeilnehmer):boolean;
 begin
 writeln('Bitte Daten eingeben oder "Ende" für Beenden');
 write('Vorname:    '); readln(Teilnehmer.Vorname);
 if (Teilnehmer.Vorname='Ende')
   then liesTeilnehmerdatenOderEnde:=false
   else begin
        write('Nachname:   '); readln(Teilnehmer.Name);
        write('Entfernung: '); readln(Teilnehmer.Entfernung);
        liesTeilnehmerdatenOderEnde:=true;
        end;
end; {function liesTeilnehmerDatenOderEnde}

begin {Hauptprogramm}
s := 1;   {s initialisieren als lfd. Nr. des unechten Elements in die Siegerliste}
sliste[1].Entfernung := 0; {Entfernung des unechten Elements eintragen}
while liesTeilnehmerDatenOderEnde(EinTeilnehmer) do
begin
if EinTeilnehmer.Entfernung = sliste[s].Entfernung
   then {füge EinTeilnehmer zu sliste hinzu, d.h. Entfernungsgleiche werden nach}
     begin                      {Zeitpunkt des Eintreffens sortiert eingetragen}
      s := s+1; {Listenzeiger um 1 weitersetzen}
      sliste[s] := EinTeilnehmer; {EinTeilnehmer eintragen}
     end
   else
   if EinTeilnehmer.Entfernung > sliste[s].Entfernung  {weiter als akt. Gewinner}
        then {lösche s, setze EinTeilnehmer als einziges neues Element aus sliste}
         begin
          s:=1; sliste[s] := EinTeilnehmer;
         end;
end; {while}
writeln('Siegerliste:');
for j:=1 to s do writeln(sliste[j].Vorname,' ',sliste[j].Name);{AusgabeSiegerlste}

end.{Hauptprogramm}
```

2.3 Variablen und Datentypen

Eine Variable besteht aus den drei Komponenten Name, Inhalt und Datentyp:

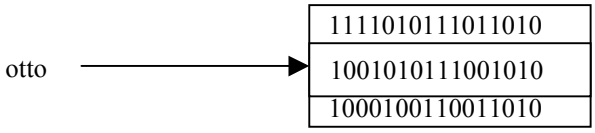
	Beispiel 1	Beispiel 2	Beispiel 3	Beispiel 4
Name	otto	theo	MeinName	ErsterBuchstabe
Inhalt	3	3. 5	‚otto'	‚x'
Datentyp	Integer (ganze Zahl)	Real (reele Zahl)	String (Zeichenkette)	Char (Zeichen)

2.4 Variablendeklaration

Im Deklarationsteil eines Pascalprogramms werden Name und Datentyp einer Variablen vereinbart.

```
VAR a     :  integer;  {ganze Zahl}
    B     :  real;     {reele Zahl}
  Name    :  string;   {Zeichenkette}
Eingabe   :  char;     {Zeichen}
Gefunden  :  boolean;  {Wahrheitswerte true/false}
Liste     :  array [1..10] of integer;  {Datentyp Feld}
```

Auf der Maschinenebene ist eine Variable als Adresse zu verstehen, die auf eine Speicherzelle im Speicher des Rechners zeigt.



3 Anweisungen und Kontrollstrukturen

In diesem Abschnitt werden Anweisungen und Kontrollstrukturen der Programmiersprache Pascal beschrieben, die den Programmablauf steuern.

3.1 Eingabeanweisungen

Mit den Pascal-Anweisungen read(_) und readln(_) ist es möglich, Variablenwerte einzulesen.

```
VAR a : integer;
begin
read(a);   {bewirkt, dass nach der Eingabe von a der Cursor }
           {hinter der Eingabe steht.}
readln(a); {bewirkt, dass nach dem Eingabe von a der Cursor in der}
           {nächsten Zeile steht}
end;
```

3.2 Ausgabeanweisungen

Mit den Pascal-Anweisungen write(_) und writeln(_) ist es möglich, Variablenwerte auf dem Bildschirm auszugeben.

```
VAR a : integer;
begin
write(a);   {bewirkt, dass nach der Ausgabe von a der Cursor }
            {hinter der Ausgabe steht.}
writeln(a); {bewirkt, dass nach dem Ausgabe von a der Cursor in der}
            {nächsten Zeile steht}
end;
```

3.3 Die Zuweisung

Die Zuweisung oder auch Wertzuweisung genannt ist der Befehl zur Zuweisung eines aus einem Ausdruck zu ermittelnden Werts an eine Variable. Die Syntax der Zuweisung lautet:

```
Variablenbezeichner := Ausdruck;
```

Der Ausdruck auf der rechten Seite des Zuweisungsoperators := wird zuerst ausgewertet und dann wird diese Wert an die Varaible auf der linken Seite übergeben. Der angegebene Name der Variablen muß im Deklarationsteil definiert worden sein. Der Zuweisungsoperator wird syntaktisch wie ein Zeichen behandelt. Der Datentyp des Ausdrucks auf der rechten Seite muß mit dem Datentyp der Variablen auf der linken Seite übereinstimmen.

3.4 Die bedingte Anweisung (if...then...else)

Die bedingte Anweisung ermöglicht die Ausführung verschiedener alternativer Programmblöcke in Abhängigkeit von der Auswertung eines logischen Ausdrucks. Syntax:

```
IF <Bedingung> THEN
                  BEGIN
                  Anweisungsblock Alternative 1
                  END
                ELSE
                  BEGIN
                  Anweisungsblock Alternative 2
                  END;
```

Hier darf kein Semikolon stehen!

Der sogenannte ELSE-Fall (Sonst-Fall) kann dabei entfallen. Besteht der jeweilige Anweisungsblock nur aus einem einzelnen Befehl, so kann die Schachtelung mit BEGIN und END entfallen.

3.4.1 Geschachtelte Fallunterscheidungen

Bedingte Fallunterscheidungen mit if...then...else können selbstverständlich auch geschachtelt werden, wie das folgende Beispiel zeigt.

```
IF a<30 then
           if a<20 then writeln('a ist kleiner als 20')
                   else writeln('a liegt zwischen 20 und 30')
        else
           writeln('a ist größer oder gleich 30');
```

3.4.2 Die Case-Anweisung

Die Case-Anweisung erlaubt die Auswahl genau einer von mehreren alternativen Anweisungen in Abhängigkeit vom Wert eines Ausdrucks z.B.:

```
Type farbe = (rot,blau,gruen,gelb,weiss);

     CASE farbe OF
       weiss : writeln('Weiß');
       rot   : writeln('Rot');
       gruen : writeln('Grün');
      end;
```

3.5 Wiederholungen (Schleifen)

Wiederholungen sind Kontrollstrukturen, die die wiederholte Ausführung eines Anweisungsblocks ermöglichen.. In der Programmiersprache Pascal stehen zwei Schleifen mit Abbruchbedingungen (repeat...until und while...do) und eine Zählschleife (for...to...do) zur Verfügung.

3.5.1 Die for-Schleife

Mit der for-Schleife ist die Wiederholung eines Anweisungsblocks für eine zu Beginn der Wiederholung feststehende Anzahl an aufeinanderfolgenden Werten eine sogenannten Zählvariablen möglich:

```
anfang := 1;
ende := 10;
for i:=anfang to ende do
    begin
      for j:=anfang to ende do
          begin
          write('*');
          end;
      writln;
    end;
```

Die Zählvariablen der for-Schleifen können innerhalb des Schleifenrumpfes benutzt werden, sollten dort aber auf keinen Fall in ihrem Wert verändert werden.

```
Var a : array [1..10] of integer;
    i : integer;
for i:=1 to 10 do begin
                  writeln(a[i]);
                  end;
```

3.5.2 Die repeat-Schleife

Die repat...until-Schleife ist eine Schleife, bei der der Schleifenrumpf zuerst durchlaufen wird, bevor die Abbruchbedingung überprüft wird. Deshalb erfolgt bei dieser Schleife mindestens ein Durchlauf des Schleifenrumpfes.

```
Repeat
  Writeln('Hauptmenü');
  Writeln('(E) Eingabe der Daten');
  Writeln('(A) Ausgabe der Daten');
  Writeln('(B) Berechnung der Funktion');
  Writeln('(X) Programm beenden);
  Readl(eingabe);
  CASE eingabe of
     'E' : Eingabe;    {Prozeduraufrufe}
     'A' : Ausgabe;
     'B' : Berechnung;
  end; {case}
until eingabe='X';
```

3.5.3 Die while-Schleife

Die while...do-Schleife ist eine Schleife, bei der zuerst die Abbruchbedingung überprüft wird, bevor der Schleifenrumpf durchlaufen wird. Deshalb kann in Abhängigkeit von der Bedingung bei dieser Schleife ein Durchlauf des Schleifenrumpfes auch gar nicht erfolgen.

```
While x<y do
  begin
    y := y-1;
    x := x-1;
  end;
```

Aufgabe: Für welche Werte von x und y arbeitet das Programmstück korrekt?

4 Selbstdefinierbare Datentypen

4.1 Der Aufzählungstyp

Mit dem Schlüsselwort TYPE kann man einen Aufzählungstyp definieren. (Die Deklaration von selbstdefinierten Datentypen erfolgt vor der Variablendeklaration mit dem Schlüsselwort VAR):

```
TYPE   spielfarbe = (karo,herz,pik,kreuz);
       farbe      = (rot,blau,gelb,gruen);
```

Entsprechend können dann Variablen von diesem Typ deklariert werden:

```
VAR trumpf     : spielfarbe;
  grundierung  : farbe;
```

Damit sind dann Anweisunge der folgenden Form möglich:

```
IF trumpf = herz then writeln('kontra');
```

4.2 Der Ausschnittstyp

```
TYPE wochentag   = (mon,die,mit,don,fre,sam,son); {Aufzählungstyp}
     werktag     = mon..fre;
     wochenende  = sam..son;
```

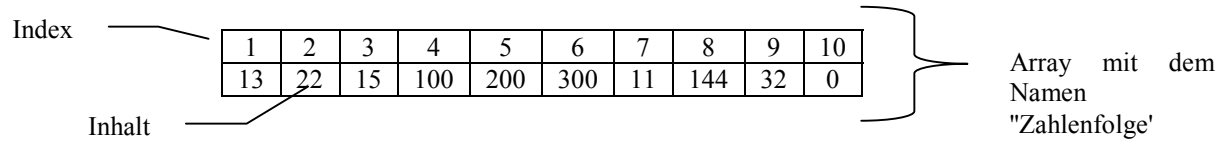
4.3 Der Set-Typ

```
TYPE zutaten     =   (zucker, mehl, sahne, honig, mandeln);
     nachtisch   = set of zutaten;
```

4.4 Der Array-Typ (Datentyp Feld)

Ein Array ist die Zusammenfassung einer festen Anzahl von Elementen des gleichen Datentyps. Die Deklarierung soll an einem Beispiel erklärt werden.

```
VAR  zahlenfolge : array [1..10] of integer;
```



Arrays können als Elemente, Objekte beliebigen Typs haben. So ist zum Beispiel folgende Deklaration möglich:

```
TYPE person = RECORD
                name     : string;
               vorname   : string;
               END;
    personenliste = ARRAY [1..100] of person;
```

4.5 Der record-Typ (Verbunddatentyp)

Der Verbunddatentyp stellt eine sehr elegante Möglichkeit dar, Informationen zu einem strukturierten Datentyp zusammenzufassen. Dadurch ist es möglich sehr komplexe Datenstrukturen zu entwickeln und komfortabel mit ihnen zu arbeiten.

```
TYPE  Zeit = RECORD
               Stunde   :  integer;
               Minute   :  integer;
             END;
```

Nach der Typdeklaration kann man verschiedene Variablen dieses Typs deklarieren.

```
VAR MeineZeit : Zeit;
    DeineZeit : Zeit;
```

Zugriff auf die Komponenten der jeweiligen Variable erhält man über den sogenannten 'Punktoperator

```
MeineZeit.Stunde := 12;
MeineZeit.Minute := 30;
```

Erleichtert wird der Umgang mit Verbunddatentypen durch die with-Anweisung.

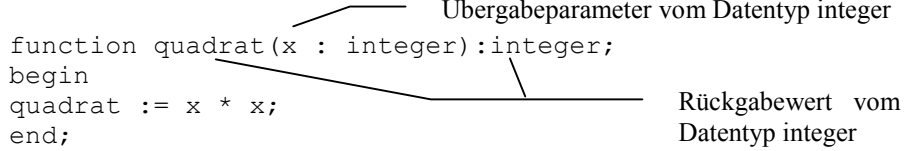
```
with MeineZeit do begin
                    Stunde := 12;
                    Minute := 30;
                  end;
```

5 Funktionen und Prozeduren

Funktionen und Prozeduren stellen Möglichkeiten dar, ein umfangreiches Programm in sinnvolle Einheiten zu unterteilen und inhaltlich zusammengehörige Vorgänge in einem Programmblock zusammenzufassen.. Programm gewinnen durch die Verwendung von Funktionen und Prozeduren deutlich an Übersichtlichkeit. Zudem verringert sich die Fehleranfälligkeit. Außerdem ist die Wiederverwendbarkeit der Prozeduren und Funktionen von großer Bedeutung. Dadurch werden Programm um vieles kürzer.

5.1 Funktionen

Eine Funktion ist ein Unterprogramm, das einen Rückgabewert liefert.



Der Aufruf der Funktion erfolgt dann mit Hilfe ihres Funktionsnamens und der Übergal

```
ErsteZahl := 2;
ergebnis := quadrat(ErsteZahl);
writeln(ergebnis);
ZweiteZahl := 4;
writeln(quadrat(ZeiteZahl));
```

5.2 Rekursive Funktionen

Eine Funktion, die sich selbst aufruft wird rekursive Funktion genannt.

```
function pot(a,n:integer):integer;
begin
if n=0 then pot := 1
       else pot := pot(a,n-1) * a;
end;
```

5.3 Prozeduren

Prozeduren sind Unterprogramme die dazu dienen können immer wiederkehrende Vorgänge zusammenzufassen oder ein größeres Programmprojekt in überschaubare Einzelteile (Module) zu zerlegen.

```
procedure hauptmenue(KundenName:string);
var eingabe;
begin
Repeat
  Writeln('Hauptmenü');
  Writeln('(E) Eingabe der Daten');
  Writeln('(A) Ausgabe der Daten');
  Writeln('(B) Berechnung der Funktion');
  Writeln('(X) Programm beenden);
  Writeln('Sehr geehrte/r Frau/Herr ',Kundenname,' bitte wählen Sie!');
  Readl(eingabe);
  CASE eingabe of
     'E' : Eingabe;
     'A' : Ausgabe;
     'B' : Berechnung;
  end; {case}
until eingabe='X';
end; {procedure hauptmenue}
```

Prozeduren werden über ihren Prozedurnamen aufgerufen. Angenommen die Prozeduren 'Begruessung', 'Verabschiedung' und die oben dargestellte Prozedur 'Hauptmenue' seien bereits deklariert, dann kann das Hauptprogramm wie folgt sehr übersichtlich aussehen.

```
begin {Hauptprogramm}
Begruessung;
hauptmenue(Name);
Verabschiedung;
end. {Hauptprogramm}
```

5.4 Rekursive Prozeduren

Eine rekursive Prozedur ist eine Prozedur, die sich selbst aufruft.

A

Abbruchbedingung · 6
Adresse · 5
Algorithmus · 3
Anweisungen · 3
Anweisungteil · 3
Array-Typ · 8
Aufzählungstyp · 7
Ausdruck · 5
Ausgabeanweisungen · 5
Ausschnittstyp · 7

B

bedingte Anweisung · 6

C

Case-Anweisung · 6

D

Datentypen · 5
Deklarationsteil · 3

F

Feld · 8
for...to...do · 6
for-Schleife · 6
Funktionen · 8

G

Geschachtelte Fallunterscheidungen · 6
geschwungenen Klammern · 3
Grundlagen · 3

I

if...then...else · 6
Inhalt · 5

K

Kommentare · 3
Kontrollstrukturen · 5

M

Module · 9

P

Pascal-Befehle · 3
Pascal-Programm · 3
Programmiersprache · 3
Programmkopf · 3
Programmname · 3
Prozeduren · 9
Prozedurnamen · 9
Punktoperator · 8

R

record-Typ · 8
Rekursive Funktionen · 9
Rekursive Prozeduren · 9
repeat...until · 6
repeat-Schleife · 7

S

Schleifen · 6
Schleifenrumpf · 7
Selbstdefinierbare Datentypen · 7
Set-Typ · 8
Speicher · 5
Speicherzelle · 5
`Suche` · 9

U

Unterprogramme · 9

V

Variable · 5
Variablen · 5
`Variablenbezeichner` · 5
Variablendeklaration · 5
Verbunddatentyp · 8

W

Wertzuweisung · 5
while...do · 6
while-Schleife · 7
Wirth · 3
with-Anweisung · 8

Z

Zählvariablen · 6
Zuweisung · 5
Zuweisungsoperators · 5