The programming language PASCAL

Eine Seminararbeit im Rahmen des Proseminars
"Programmiersprachen"

Autor:

Name : SAYGILI Melek

MatNr : 0253156

Kennzahl : 534

Universität : Technische Universität Wien

E-mail : sygmelek@yahoo.de

Kurzfassung

Pascal ist eine Lehrsprache, welche leicht erlernbar ist und eine einfache Syntax hat. Zuerst werde ich Motivation zur Entwicklung der Sprache geben. Danach werde ich einen Kurzeinblick über die Syntax und Semantik geben, sowie über die Grundlagen des Pascals. Ich werde noch versuchen den Programmfluss der Sprache zu erklären. Zuletzt werde ich über die Vorteile und Nachteile des Pascals sprechen.

Motivation zur Entwicklung der Sprache

Bei der Entwicklung von PASCAL wurde von N.Wirth Anfang der 70er Jahre geschrieben. Zuvor hatte Wirth schon Erfahrungen in der Entwicklung von Programmiersprache sammeln können; er wirkte an der Entwicklung der Sprache ALGOL 60, welche als Vorgänger von PASCAL gilt. Bei Entwicklung von PASCAL wurde versucht zwei Anforderungen zu erfüllen. Zum Einen sollte das Erlernen sowohl der Sprache PASCAL, als auch der grundlegenden Konzepte auf denen PASCAL basiert, leicht fallen und selbsterklärend sein. Die andere Anforderung war es eine Sprache zu schaffen deren Implementierungen auf den damals gegenwärtigen Rechnern über eine hohe Effizienz verfügen, als auch verlässlich funktionieren. Damit sollte der damals herrschenden Meinung, dass eine nützliche Programmiersprache entweder einen langsamen Compiler verwendet, oder aber die Programme zur Laufzeit langsam laufen, entgegengewirkt werden. Außerdem sollte PASKAL die Annahme dass effiziente Sprachen immer zu gewissen Fehlern neigen widerleg werden. Da die Programmiersprache mit denen das Programmieren erlernt wird, meistens aufgrund der Verbreitung und gegenwärtigen Akzeptanz gewählt werden, und die Sprachen die gelehrt werden widerrum diejenigen sind die am Meisten verwendet werden, gerät die Entwicklung leicht ins Stocken. Dass PASCAL zum Teil als etwas völlig Neues entwickelt wurde, passiert auch um diesen Effekt entgegenzuwirken.

Da den Entwicklern von PASCAL aber auch klar war, dass bereits existierende Sprachen, insofern sie mit den Anforderungen der neu zu entwickelnden Sprache übereinstimmen, als Grundlage für die Entwicklung verwendet werden können, basiert auch PASCAL auf einer vorhergehenden Sprache; ALGOL 60.ALGOL 60 wurde als Basis für PASCAL gewählt weil es die Anforderungen der Entwickler in punkto Systemstruktur, Flexibilität sowohl beim Programmieren als auch beim Strukturieren von Daten und effizienter Implementierungsmöglichkeiten eher erfüllte als alle anderen Programmiersprachen zu dieser Zeit. Trotzdem wurde es vermieden ALGOL 60 als Teil von PASCAL zu implementieren, da vor allem im Bereich der Datenstrukturen grobe Unterschiede vorherrschten.

Mit der Einführung con file- und record- Strukturen in PASCAL wurde versucht die Entwicklung sowohl von technisch/wissenschaftlichen, als auch kommerziellen Programmen zu ermöglichen.

3)Kurzüberblick über Syntax und Semantik :

Variablen Typen und Deklarationen

Variablen sind ähnlich wie Konstanten, aber ihre Werte können geändert werden, wenn das Programm läuft. Variablen müssen im Pascal zuerst deklariert werden, bevor sie verwendet werden können.
IdentifierList ist eine Reihe vom Bezeichnen, getrennt durch Kommas (,). Alle Bezeichnen in der Liste muss vom gleichen Datentyp sein.

In der Sprache PASCAL gibt es 5 Basisdatentypen die für Variable verwendet werden können, sie werden auch skalare Standardtypen genannt.

- **integer:** wird für die positive und negative ganze Zahlen verwendet. Wie groß der Bereich der Zahlen ist den die variable annehmen kann, kommt auf das System in dem PASCAL implementiert ist an.
- **boolean:** entspricht mit einem Bit; nur zwei mögliche werte: false, true
- **char:** entspricht einem „Zeichen“, Ziffern, Buchstaben und Sonderzeichnen. Wie viele und welche Zeichen ein Charakter annehmen kann, hängt vom System ab.
- **alfa :** entspricht einer Zeichenkette mit der Länge N.N hängt Widerrum vom System ab.

Variablen eines Standardtyps werden wie folgt deklariert:

```
a,b : integer ;  bl: boolean;
```

Neben diesen Standardtypen gibt es auch noch die Möglichkeit strukturierte Datentypen zu verwendet. Zu diesen zählen:

- **Array:** ein Array ist ein strukturierte Datentyp der eine bestimmte Anzahl von Elementen gleichen Typs aufnehmen kann. Angesprochen werden die Elemente über einen Index. Es können auch mehrdimensionale Arrays erstellt werden (Arrays mit mehr als einem Index). z.B.:

  ```
  x = array [1..100] of integer;
  y = array[1..5,1..5]of integer;
  ```

  Auf die Elemente des Array lässt sich folgendermaßen zugreifen:

  ```
  x[20]
  y[3,3]
  ```

- **Record**: Ein Record besteht aus einer fixen Anzahl von Elementen, welche allerdings unterschiedlichen Typen sein können. Somit lassen sich mehrere Variablen die logisch zusammenhängen leicht unter einem Namen abgespeichert.z.B.:
  Typdefinition:

  ```
  Auto = record   name,farbe: alfa;
                  baujahr, kaufpreis, kilometerstand: integer;
           end
  Complex = record realTeil, imaginaerTail: real;
            end
  ```

  Deklaration:

  ```
  a,b: Complex;
  renault5, mazda323: Auto;
  ```

  Zugriff auf die Komponenten der Variablen

  ```
  Renault5.kaufpreis
  a.realTeil
  ```

- **Powerset:** Der Typ Powerset (Mengentyp) definiert eine Menge von Werten als den Powerset eines anderen skalaren Typen, dem so genannten Basistyp. Auf das so genannte Powerset lassen sich auf dann folgende Operationen anwenden: Union, intersection, set difference und membership.
- **File:** Die File ist ein Datentyp bestechend aus Komponenten gleichen Typs, die Anzahl der Komponenten muss bei der Definition allerdings nicht festgelegt werden. Somit verfügt die Struktur File über eine variable Länge. Jede File hat einen von 3 möglichen Zuständen: input (aus der File wird gelesen), output(auf die File wird geschrieben) und neutral. Jede Variable des Typs File erhält auch

einen so genannten file pointer, mit welchem sich auf die einzelnen Elemente zugreifen lässt. Der Zugriff auf diesen file pointer findet mit Standardprozeduren statt.

Textdatei = **file of** char;

Es lässt sich immer nur das Element auf welches der File Pointer Zeigt ansprechen.

- **Class:** In eine Klasse lassen sich während der Laufzeit des Programmes beliebig viele Elemente eines Datentyps mit der Standardprozedur alloc schreiben. Die maximale Anzahl von Elemente muss allerdings in der Variablen Deklaration angegeben werden.z.B:
  - Parkplatz: **class** 500 of Auto;
- **Pointer**: Eine Pointervariable lässt sich mit einer beliebigen Klasse „verbinden“ und ermöglicht so den Zugriff auf die Elemente der Klasse. Außerdem gibt es eine Pointerkostante, nil, welche auf keine Komponente zeigt.
  - P1: |parkplatz;
  - P1|.renault5|.kilometerstand

Operatoren

Für Operatoren gibt es in PASCAL 4 verschiedene Prioritätsklassen- die oberste Priorität hat der Operator ¬ , danach die Multiplikationsoperatoren, die Additionsoperatoren und am Ende der Prioritätskette stehen die sogenanten relationalen Operatoren.

Der ¬ Operator

Der ¬ Operator angewandt auf ein Variable vom Typ Boolean bedeutet Negation der Variable.

Multiplikationsoperatoren

operator	operation	Type of operands	result
*	multiplikation	Real or integer	Integer,if both operands are of type integer, real otherwise
/	division	Real or integer	real
div	Division mit truncation	integer	Integer
mod	M mod n=m-((m div n)*n)	integer	Integer
^	Logische „and“ set intersection	Boolean Any Powerset type T	Boolean T

(WIRTH,1971,48)

Additionsoperatoren

operator	operation	Type of operands	result
+	Additon	Real or integer	Integer,if both operands are of type integer, real otherwise
-	Subtraction	Real or integer	real
-	Set difference	Any Powerset type T	T
∨	Logische „or“ set union	Boolean	Boolean
		any Powerset type T	T

Relationale Operatoren

operator	Type of operands	result
=	any type,except file and class types	boolean
< >	any scalar or	boolean
=< >=	subrange type	boolean
in	any scalar or subrange type and its powerset type reseptively	boolean

(WIRTH, 1971, 49)

- **Reservierte Namen in Pascal**
  Einige Bezeichner werden im Pascal als syntaktische Elemente aufgehoben. Ihnen werden nicht erlaubt, diese als Ihre Bezeichner zu verwenden. Diese schließen ein, aber werden nicht begrenzt:

and	arry	begin	case	const	div	do	downto
else	end	file	for	funktion	goto	if	in
label	mod	nil	not	or	paced	procedure	program
record	repeat	set	then	to	type	until	var
while	with						

- **Standard Funktionen**
  Funktionen werden benannt, indem
  man den Funktion Namen verwendet, der in Klammern von den Argumenten gefolgt wird.
  Standardfunktionen in Pascal sind:

  sin(x) sqrt(x) arctan(x) trunc(x) round(x) odd(x) cos(x)
  abs(x) sqr(x) exp(x) ln(x)

Statements

Statements zeigen algorithmische Abläufe, und sind in jedem Fall ausführbar.

Einfache Statements

- **Assignment Statement(Zuweisung)**
  Bei einer Zuweisung wird der Wert einer Variable gesetzt, hat die Variable bereits einen Wert, so wirt der alte Wert überschreiben. Die Variable und der Ausdruck der den neuen Wert liefert, müssen vom selben Typ sein, außer die Variable ist vom Typ real und der Ausdruck liefert Integer oder der Ausdruck liefert einen Wert dessen Typ ein Teilbereich des Variablentyps ist.z.B.:
  Sgrt(5)
  Transpose(a,n,m)
- **Gotto Statement(Sprunganweisung)**
  Mithilfe des gotto Statements kann man an definierte Punkte im Quellentext „springen“. Diese Punkte müssen vorher, zum Beispiel in einem Compound- Statement definiert werden. Zu beachten ist dass sich der Punkt zu dem man „springen“ will innerhalb der gleichen Prozedur befinden muss.

Strukturierte Statements

Strukturierte Statements bestehen aus mehreren Statements, welche entweder gleichzeitig, abhängig von einem anderen Ausdruck oder sich wiederholend (abhängig von einem anderen Ausdruck) ablaufen.

- **Compound Statements (Zusammengesetzte Anweisung)**

  Hier werden in der Reihenfolge wie sie geschrieben werden ausgeführt. Außerdem kann in diesem Statement ein Punkt gesetzt werden an dem man mithilfe des Goto Statements hin „springen“ kann.
  begin x:=y; y:=z **end**
- **Conditional Statements (abhängige Statements)**
  Es gibt zwei Typen von abhängigen Statements. Die if- Anweisung und die case- Anweisung.
  - **if-Statement:**

Bei diesem Statement wird aufgrund eines Ausdruckes (des Typs Boolean) entscheiden ob bestimmte Anweisungen ausgeführt werden sollen oder nicht. Bei positivem Ausdruck wird der if- Zweig des Statements ausgeführt bei einem negativen Ausdruck der else- Zweig.z.B:

```
if  x<1.4 then x:=x+3 else x:=x-3
```

- **case- Statement**

  Das case- Statement ist dem if- Statement sehr ähnlich, allerdings ist der Ausdruck aufgrundessen entscheidet wird nicht vom Typ Boolean, somit gibt es mehr als nur zwei mögliche Zweige die ausgeführt werden. z.B:

```
case romanNummer of
i: arabicNummer :=1;
v: arabicNummer :=5;
x: arabicNummer :=10;
end
```

- **Repetitive Statements(wiederholende Statements)**

  Eine oder mehrere Anweisungen werden, abhängig von einer bedingung, widerholt. Aufgrundessen wird dieser Typ Statements auch Schleife genant.

  - **While- Schleife**

    Bei dieser Schleife wird die Entscheidung ob der Ausdruck noch einmal ausgeführt werden soll am Anfang der Schleife getroffen. Ist die Bedingung erfühlt werden die Anweisung innerhalb der Schleife noch einmal ausgefühlt, ist dem nicht so wird die Schleife kein weiteres Mal duchlaufen.z.B:

```
while (a<=4) do a:=a+1
while (a<b) do
begin
        a:= a/3; b=b/2
end
```

  - **repeat- Schleife**

    Die repeat- Schleife entspricht vom der Funktion her der while Schleife- allerdings wird die Entscheidung ob die Ausdrücke nochmals ausgeführt werden am Ende der Schleife getroffen. Gegensatz zur while- Schleife: ist der Entscheidungsausdrucks wahr wird die Schleife abgebrochen.z.B:

```
repeat  a:=a+1 until (a>5)
repeat  a:=a/4 ;
        b:=b/2 ;
until (a>=b)
```

  - **for- Schleife**

    Die for-Schleife wird von einer Variable gesteuert, als erstes erhält diese Variable einen Startwert, dieser Variable wird bei jedem Schleifenduchlauf erhöht und die Schleife wird beendet wenn die Variable einen vordefinierten Zielwert erreicht hat.z.B:

```
for  i:=0  to  99   go erg:=erg *i
```

- **with-Statement**

  Hierbei handelt es sich um keine Schleife. Das with- Statement dient dazu Rekordkomponenten ansprechen tu können ohne den Bezeichner der Records angeben zu müssen.

```
wiht renault5 do
begin
        kilometerstand:=0; baujahr:=2005; kaufpreis:=5500;
        name:=Renault5TURBO; farbe:=rot;
end
```

Prozeduren und Funktionen

Prozeduren sind Teile von Programmen die mit einem Identifier gekennzeichnet werden, um an einer anderen Stelle im Programm mit einem Prozeduraufruf ausgeführt zu werden. Eine Prozedur besteht im Regalfall aus folgenden Teilen:

⟨procedure declaration⟩ ::=
⟨procedure heading⟩
⟨constant definition part⟩ ⟨type definition part⟩
⟨variable declaration part⟩
⟨procedure and function declaration part⟩ ⟨statement part⟩

(WIRTH, 1971, 55)

Der Prozedurkopf(<Procedure heading>) und der Anweisungsteil (<statement part>) dürfen nicht leer sein, ansonsten ist es keine gültige Prozedur und kann somit auch verwendet werden.

Im Prozedurkopf wird ein Name für die Prozedur und die zu übergebenden Parameter vereinbart. In den nächsten Teilen lassen sich Konstanten (<constant definition part>), Datentypen (<type definition part>), Variablen (<variable declaration part>) definiert. Alles was an diesem Punkt der Prozedur definiert wurden hat allerdings einen eingeschränkten Geltungsbereich- die Prozedur selbst. Somit kann von „außerhalb“(z.B. eine andere Prozedur) nicht auf die Variablen in der Prozedur zugreifen werden, deshalb wird alles was hier definiert ist acuh lokal gennant. Der Anweisungsteil (<statement part>) ist wohl der wichtigste Teil der Prozedur hier stehen die Anweisungen die ausgefühlt werden sollen wenn die Prozedur aufgerufen wird.

Beispiel für eine Prozedur:

```
procedure readinteger (var x: integer);
    var i, j: integer;
begin i:=0;
    while (input_>='0')^(input_<='9') do
    begin j:=int(input_)-int('0');
        i:=i*10+j;
        get(input)
    end
    x:=i
end
```

(WIRHT, 1971,56)

Funktionen sind Prozeduren sehr ähnlich, allerdings liefern Funktionen einen Wert „zurück“, d.h. das Programm welches die Prozedur aufruft erhält einen Wert von der Prozedur. Welchen Typ dieser Rückgabewert hat kann vom Programmierer selbst bestimmt werden. Die Struktur einer Funktion ist gleich wie die einer Prozedur, der einzige Unterschied liegt Funktionskopf; hier muss bei einer Funktion der Typ des Rückgabewerts gewählt werden.

function Sqrt(x:real): real;

In der Funktion existiert eine Variable vom Typ real, welche unter dem Namen Sqrt angesprochen werden kann. Der Wert den diese Variable am Ende der Funktion hat wird zurück geliefert.

Besonderheiten und Anwendungsbeispiele

Strukturierte Datentypen

Eine große Besonderheit von PASCAL gegenüber der anderen Programmiersprache der damaligen zeit war die Möglichkeit Datenstrukturen zu definieren. Wie bereits im vorherigen Kapitel behandelt. Wird dies möglich durch den Datentyp record, mit im es möglich mehrere, verschiedene, aber logisch zusammen gehörige Datentypen und einem Namen abzuspeichern. Dadurch lassen sie viele komplexe Probleme, sowohl im wissenschaftlichen als auch im kommerziellen bereich einfach und effizienter lösen, vor allem deswegen weil große Programme besser strukturiert werden können und somit der Code einfache les- und wart- bar wird.

Pointer

Mit PASCAL wurden auch pointervariablen eingeführt. Mit ihnen ist es möglich große mengen von record-variablen(welche zusammengefasst in einer Klasse liegen)zu verwalten, zu organisieren. Dies ist vor allem bei Anwendungen welche mit großen Datenmengen arbeiten vom großem nutzen.

Powerset

Der Typ Powerset wurde ebenfalls von PASCAL eingeführt. Hier mit ist es möglich mengen zu definieren(unter mengen von anderen Datentypen)auf die dann die klassischen Mengenoperationen angewendet werden können

Erlernbarkeit

Da die Grundkonzepte die hinter der Sprache PACAL stehen (z.B. Strukturierung von Daten) durch die zu verwendende Syntax repräsentiert werden, ist PASCAL ideal um das Programmieren zu erlernen. PASCAL ist aufgrund dieser Verständlichkeit nicht nur geeignet um PASCAL selbst leicht zu erlernen, sondern auch um grundlegende Konzepte und ablaufe in Programmen leicht zu verstehen.
Deswegen fand Pascal überall Anwendung wo es darum galt Programmierer aus- und weiter zu bilden
Zusammenfassung vor- und nachteile

Vorteile

- relativ höhe Geschwindigkeit
- gute Erlernbarkeit
- viele Möglichkeiten bei der Beschreibung und Strukturierung von Daten
- große Anwendungsbereich sowohl in wissenschaftlichen/kommerziellen Bereich

Nachteile

- nicht mehr zeit gemäß
- unterstütz nicht aller Konzepte der Objektorientierten Programmierung

Litarathurliste

- Wirth N., The programming language Pascal, Acta Informatica 1, Seite 35-61,1971
- Böhme P., Progtammiersprache Pascal, http://www.db.informatik.uni-kassel.de/Help/pascal/einfuerung/index.html, 1966
- Hopfer Informationsverarbeitung mit Pascal