Programiersprache Oberon

"Make it as simple as possible but not simpler"

Grundlagen Wissenschaftlichen Arbeitens

TU 185.165 J.Knoop Programmiersprachen,

Mustafa Söylemez
E0327329

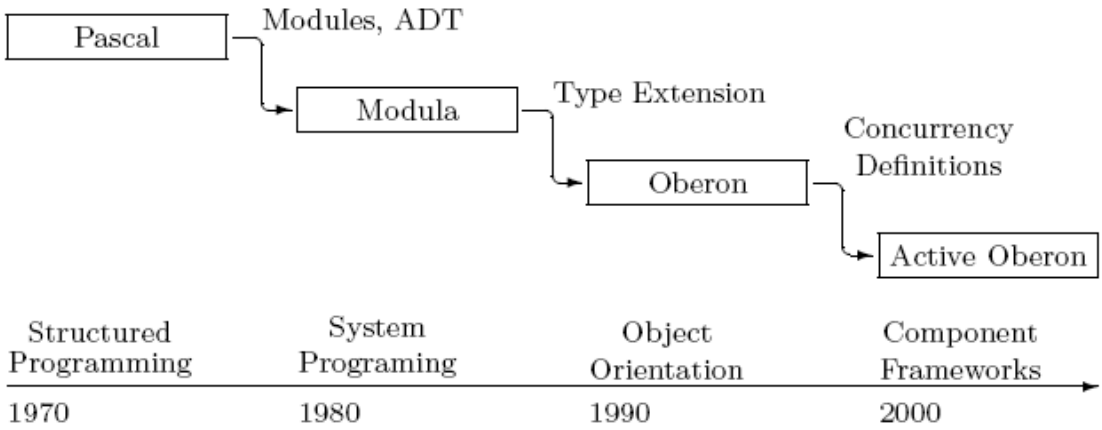
Inhalt

Einführung

Oberon ist eine universelle (objektorientierte) Programmiersprache, die von Niklaus Wirth und Jürg Gutknecht entwickelte.
Neben der Programmiersprache kann Oberon auch als Betriebssystem (Native Oberon) oder als Programm-Entwicklungsumgebung genutzt werden.

1. Familien Entwicklung

Die Sprache Oberon ist der neueste Nachkomme der ALGOL-, Pascal- und Modula-Familie



Die Oberon Sprachce Familien Entwicklung

- Pascal wurde als Sprache begriffen, um kleine Programme auszudrücken; seine Einfachheit und Magerkeit bildeten es besonders gut angepaßt für die Unterrichten Programmierung

- Modula entwickelte vom Pascal als Sprache für die Systemprogrammierung und profitierte von der praktischen Erfahrung, die während der Entwicklung der Workstation Lilith und des Betriebssystems Medos gewonnen wurde

- Die Notwendigkeit, die Programmierung im großen Paradigma zu stützen war der Beweggrund für Oberon. Die Betriebssystemprojekte Oberon wurden parallel zu der Sprache entwickelt und gelassen die Sprachverbesserungen prüfen und auswerten

- Viele experimentelle Sprachverlängerungen sind für Oberon .
  Object Oberon , Oberon-2 und Froderon vorgeschlagen worden, das weitere objektorientierte Eigenschaften der Sprache hinzufügend erforscht wird; Oberon-V schlug Hinzufügung für parallele Betriebe auf vektorcomputern vor; Oberon-XSC addierte mathematische Eigenschaften, um wissenschaftliche Berechnung zu stützen; das Moduleinbetten wurde auch vorgeschlagen.

- Aktives Oberon ist der erste Exponent eines neuen Erzeugung von Sprachen in dieser Familie. Unser Beweggrund soll die Parallelität und Bestandteil stützen, die in der Sprache in einer sauberen, nahtlosen Weise modellieren

Warum Oberon..?

Die antwort ist, weil die sprache oberon auf relativ wenigen, aber fundamentalen konzenten aufbaut, sie ist streng strukturiert weil sie auf modernen Rechnern effizient implementiert werden kann.

2. Oberon syntax

Programmieren bedeutet das Erstellen neuer Programme. Zu diesem Zweck ist nur eine präzise Beschreibung angemessen.
Eine Sprache ist eine unendliche Menge von Sätzen, deren Aufbau bestimmten Vorschriften, der Syntax, genügt.
In Oberon heißen diese Sätze Übersetzungseinheiten. Jeder Satz besteht aus einer endlichen Folge von Symbolen,die aus einem endlichen Vokabular stammen.
Die menge der Regeln ist die grammatik der sprache und beschreibt deren Syntax.
Das programm besteht aus teilen, die syntaktische Einheiten genannt werden.
Beispiele sind deklarationen, Anweisunfen oder Ausdrücke.

3. Das vocabular und Darstellung

3.1 Namen

ident = letter { letter | digit }.

Eine Name (identifier) besteht aus einer Folge von Buchstaben (letter) und Dezimalziffern (digit) die immer mit einem Buchstaben beginnt. Groß und kleinbuchstaben werden unterschieden.

- Beispiele fur Zeichenfolgen, die gültige Namen gültigen, sind:

  List list a12

- Beispiele fur Zeichenfolgen, die keine gültigen Namen gültigen, sind:

  Leerzeichen, Bindestrich, Unterstreichungszeichen sind nicht erlaubt. Und die ertes Zeichen muss ein Buchtabe sein.

3.2 Zahlen

Number = integer | real .

Zahlen sind entweder vorzeichenlose ganze Zahlen (integer) oder vorzeichenlose reelle Zahlen (real).

Beispiele für gültige Zahlen sind:

1987
100H = 256
12.3
4.567E8 = 456700000
0.57712566D-6 = 0.00000057712566

3.2 Zeichenkonstanten

CharConstant = “ “ “ character “ “ “ | digit {hexDigit} “X“.

Zeichenkonstanten (character Constants) werden entweder durch ein einzelnes Zeichen in Anführungszeichen oder durch die hexadezimale Ordnungszahl des Zeichens, gefolgt vom Buchstaben X, bezeichnet.

Beispiele fur zeichenkonstanten sind

“a“ , “b“, “1“

3.3 Zeichenketten

string = “ “ “ {character} “ “ “.

Zeichenketten (strings) sind in Anführungszeichen (“) eingeschlossene Zeichenfolgen. Eine Zeichenkette kann selbst
keine Anführungszeichen enthalten. Die Anzahl der Zeichen einer Zeichenkette wird als ihre Länge bezeichnet. Zeichenketten
können an Zeichen-Arrays zugewiesen und mit diesen verglichen werden.

“OBERON“ , “das ist eine zeichenkette“

3.4 Operatoren

Operators und Begrenzer (Delimiters) sind die nachfolgend aufgeführten Spezialzeichen, Zeichenpaare und reservierten Wörter. Die reservierten Wörter, auch Schlüsselwörter genannt, bestehen ausschließlich aus Großbuchstaben und können nicht als Namen benutzt werden

+	:=	ARRAY	IS	TO
-	^	BEGIN	LOOP	TYPE
*	=	CASE	MOD	UNTIL
/	#	CONST	MODULE	VAR
~	<	DIV	NIL	WHILE
&	>	DO	OF	WITH
.	<=	ELSE	OR	
,	>=	ELSIF	POINTER	
;	..	END	PROCEDURE	
|	:	EXIT	RECORD	
(	)	IF	REPEAT	
[	]	IMPORT	RETURN	
{	}	IN	THEN	

4. Deklarationen und Sichtbarkeitsregeln

Jeder in einem Programm verwendete Name muss mittels einer Deklaration eingeführt werden,in oberon mussün alle namen vor ihrer benutzung deklariert werden.

Deklarationen dienen auch dazu, bestimmte unveränderliche Eigenschaften eines Objekts festzulegen, zum Beispiel ob es sich um eine Konstante, einen Typ, eine Variable oder eine Prozedur handelt.

(die Deklarationen führt ein und definiert ihren Typ.)

Folgende Namen sind vordeklariert:

ABS	COPY	INC	LONGREAL	REAL
ASH	DEC	INCL	MAX	SET
BOOLEAN	ENTIER	INTEGER	MIN	SHORT
CAP	EXCEL	LEN	NEW	SHORTINT
CHAR	FALSE	LONG	ODD	SIZE
CHR	HALT	LONGINT	ORD	TRUE

4.1 Konstantendeklarationen

Eine Konstantendeklaration bindet einen Namen an einen konstanten Wert.

Beispiele :

CONST a = 16807 ; N = 100 ; m= 2147483647 .

4.2 Variablendeklarationen

Variablendeklarationen führen Variablen ein und binden einen Namen und einen Typ daran.

VariableDeclaration = IdentList ":" type .

An dieser Stelle ist type noch auf die vordeifinierten Namen der Grundtypen SHORTING ; INTEGER , LONGINT; REAL; LONGREAL;BOOLEAN;CHAR und SET beschränkt.

LONGREAL ⊇ REAL ⊇ LONGINT ⊇ INTEGER ⊇ SHORTINT

Beispiele :

Var
i, m, n: INTEGER; index: LONGINT,
ch: char

4.3 Typdeklarationen

Eine Typdeklaration legt die Menge der Werte fest, die Variablen dieses Typs annehmen können, und definiert die darauf anwendbaren Operationen. Eine Typdeklaration bindet einen Namen an einen Typ, der entweder unstrukturiert sein kann (Grundtyp) oder strukturiert. Im letzteren Fall definiert er auch die Struktur von Variablen dieses Typs und, implizit,die Operationen, die auf die Komponenten angewendet werden können.

TypeDeclaration = identdef "=" type .
type = qualident | ArrayType | RecordType | PointerType | ProcedureType .

Beispiele :

Table = ARRAY N OF REAL
Tree = POINTER TO Node
Node = RECORD key: INTEGER;
left, right: Tree

4.3 Array-deklarationen

Ein Array ist eine Struktur, die aus einer festen Anzahl von Elementen desselben Typs, ***Elementtyp*** genannt, besteht. Die Anzahl der Elemente eines Arrays heißt seine *Länge.* Die Elemente eines Arrays werden durch Indizes bezeichnet, die ganze Zahlen zwischen 0 und der Länge minus 1 sind.

ArrayType = Array length {“,“ length} OF type
length = ConstExpression .

Beispiele :

VAR v: ARRAY 3 OF REAL;

(im folgenden beispiel einer Array-deklarationen besteht die Variable v aus 3 Elementen,alle vom Typ REAL.)

5. Operatoren

Die Syntax von Ausdrücken unterscheidet zwischen vier Klassen von Operatoren mit unterschiedlicher Bindungsstärke.Der Operator ~ hat die höchste Bindungsstärke, gefolgt von Multiplikationsoperatoren, Additionsoperatoren und Relationen.

Manchmal werden verschiedene Operationen durch das gleiche Symbol bezeichnet. In diesen Fällen wird die tatsächliche peration durch den Typ der Operanden bestimmt.

5.1 Logische Operatoren

Diese Operatoren sind auf Operanden vom Typ BOOLEAN anwendbar und liefern ein Ergebnis vom Typ BOOLEAN.

Symbol	*Ergebnis*
OR	logische Disjunktion
&	logische Konjunktion
~	Negation

5.2 Arithmetische Operatoren

Die Operatoren +, -, * ,/ sind auf alle numerischen DIV,MOD nur auf ganzzahlige typ anwendbar.

Symbol	*Ergebnis*
+	Summe
-	Differenz
*	Produkt
/	Quotient
DIV	ganzzahliger Quotient
MOD	ganzzahliger Rest

5.3 Mengen- Operatoren

Mengen sind Werte vom Typ SET. Mengenoperatoren sind auf Operanden dieses Typs anwendbar.

Symbol	*Ergebnis*
+	Vereinigung
-	Differenz
*	Durchschnitt
/	symmetrische Differenz

5.4 Relationen

Relationen liefern ein Ergebnis vom Typ BOOLEAN. Die Ordnungsrelationen <, <=, > und >= sind auf numerische Typen, CHAR und Zeichen-Arrays (Zeichenketten) anwendbar.

Symbol	*Relation*
=	gleich
#	ungleich
<	kleiner
<=	kleiner oder gleich
>	größer
>=	größer oder gleich
IN	Mengenzugehörigkeit
IS	Typtest

6. Kontrolstrukturen

Eine Berechnung ist im wesentlichen eine Folge von Aktionen, die hintereinender ausgeführt werde.
Die Abfolge der Aktionen wird von Kontrollanweisungen festgelegt, die bedingte Ausführung ,Auswahl oder Wiederholung von Anweisungen oder ganzen Anweisungsfolgen spezifieren.
Sprachen mit srtuktierten Anweisungen werden als strukturierte Sprachen bezeichnet. Jahrelange Erfahrung werden als strukturierte Sprachen bezeichnet. Jahrelange Erfahrung hat gezeigt, dass geeignete Kontrollanweisungen Hand in Hand gehen mit effizienter Programmentwicklung – erst dadurch werden die Programme lesbar und vertrauenswürdig.

6.1 If-Anweisungen

IF-Anweisungen spezifizieren die bedingte Ausführung von Anweisungen. Der boolesche Ausdruck vor einer bedingten Anweisung wird *Wache* (guard) genannt. Die Wachen werden in der Reihenfolge ihres Auftretens ausgewertet, bis eine davon den Wert TRUE ergibt; die damit verbundene Anweisungsfolge wird ausgeführt. Ist keine der Wachen erfüllt, so wird die Anweisungsfolge hinter dem Symbol ELSE ausgeführt, sofern diese vorhanden ist.

IF x < 0 THEN y:= -1

ELSEIF x > 0 THEN y:= 1

ELSE y:= 0

End

6.2 Case-Anweisungen

Eine If-Anweisung mit mehreren ELSEIF-Zweigen erlaubt die Auswahl einer Anweisungsfolge unter der Kontrolle von mehreren Bedingungen.

IF-Anweisung :

IF res = 0 THEN Out.String (“renamed”)

ELSEIF res = 1 THEN Out.String(“name existed already”)

ELSEIF res = 2 THEN Out.String(“name does not exist”)

ELSEIF res = 3 THEN Out.String(“System error”)

END;

CASE-Anweisung :

CASE res OF

0: out.String(“renamed”)

1: out.String(“name existed already”)

2: out.String(“name does not exist”)

3: out.String(“System error”)

6.3 While-Anweisungen

While-Anweisungen spezifizieren die Wiederholung einer Anweisungsfolge. Wenn ein boolescher Ausdruck (guard) TRUEergibt, wird die Anweisungsfolge ausgeführt. Das Auswerten des Ausdrucks und das Ausführen der Anweisungsfolge werdenso lange wiederholt, wie der Ausdruck TRUE ergibt.

```
j := 0

WHILE j < n DO

…; j := j + 1

END;
```

Da die Varriable j, auch Kontrollvariable genannt, von 0 bis n-1 zählt, nennt man obige Konstruktion eine Zählschleife. Die While-Anweisung ist aber nicht auf Zählschleifen beschränkt,sondern kann auch benutz werden,wenn nicht von vorneherein klar ist, wie oft eine Schleige durchlaufen werden soll.

6.4 Repeat-Anweisung

Repeat-Anweisungen spezifizieren die wiederholte Ausführung einer Anweisungsfolge bis eine Bedingung erfüllt ist. DieAnweisungsfolge wird mindestens einmal ausgeführt.

RepeatStatement = REPEAT StatementSequence UNTIL expression

```
j :=0 ;
REPEAT
        … ; j := j+1
UNTIL j = n ;
```

6.5 Loop-Anweisung

Eine Loop-Anweisung spezifiziert die wiederholte Ausführung einer Anweisungsfolge. Sie wird durch Ausführung einerExit-Anweisung innerhalb der Anweisungsfolge beendet.

LoopStatement = LOOP StatementSequence END .

Ein Beispiel ist:

```
LOOP
IF t1 = NIL THEN EXIT END;
IF k < t1.key THEN t2 := t1.left; p := TRUE
ELSIF k > t1.key THEN t2 := t1.right; p := FALSE
ELSE EXIT
END;
t1 := t2
END
```

Eine Beispiel vom Oberon…:

```
MODULE dezimal;
IMPORT Out;
VAR i: LONGINT;
BEGIN
FOR i:=0 TO 9 DO
Out.LongInt(i,2);END(*FOR*);
Out.String('  end'); Out.Ln;
END dezimal.
```

Die Programme ausführt : 0 1 2 3 4 5 6 7 8 9 end

Bisher wurden zwei wichtige Begriffe einführt :

1) Deklarationen, die einen Namen an einen Type oder an einen Wert binden;

2) Anweisungsfolgen einschließlich Kontrollanweisungen, die Algorithmen beschreiben

jetzt verbindet nun Deklarationen und und Anweisungsfolgen durch Einführung von textuellen Klammern, nämlich Prozeduren und Modulen.

7. Prozeduren und Module

Prozeduren ihrer einfachsten Form als benannte Anweisungsfolgen betrachtet werden. Ein Modul ist eine Textuelleklammer die konstanten und Variablendeklarationen sowie eine Anzahl von Prozeduren umführen. Modul ist aber auch die Einheit, die vom Compiler akzeptiert wird. Übersetzte Module, Objekt-Module genannt,können in der Bibliothek eines Rechners abgelegt und zur Ausführüng in den Speicher geladen werden.

Betriebssysteme bieten dem Benutzer die Möglichkeit, in der Bibliothek eines Rechners abgelegten Programmcode auszuführen. In Oberon ist die Grundheit der Programmausführung die Prozedur. Das steht im Gegensatz zum traditionellen Konzept eines Hauptprogramms als kleinste ausführbare Einheit.

Zusätzlich zu den Formalparametern und lokal deklarierten Objekten sind auch die Objekte aus der Umgebung der Prozedur innerhalb der Prozedur sichtbar (außer jenen Objekten, die durch lokal deklarierte Objekte mit gleichem Namen verdeckt werden). Das Aufrufen einer Prozedur innerhalb ihrer eigenen Deklaration bedeutet einen rekursiven Prozeduraufruf.

Sowohl Produren als auch Module spilen eine wichtige Rolle bei der Strukturierung komplexer Programme.

Ein Modul ist eine Sammlung von Konstanten-, Typen-, Variablen-und Prozedurdeklarationen und eine Anweisungsfolge zum Zweck der Initialisierung der globalen Variablen. Ein Modul stellt typischerweise einen Text dar, der als Einheit übersetzt werden kann.

module = MODULE ident ";"

[ImportList]

DeclarationSequence

[BEGIN StatementSequence]

END ident "." .

Eine Beispiel :

```
MODULE proc1;
IMPORT Out;
VAR x, y:LONGINT;

PROCEDURE add1 (a:LONGINT; VAR b:LONGINT); BEGIN b:=a+1; END add1;

BEGIN x:=1; add1(x,y); Out.LongInt(x,4);Out.LongInt(y,4);Out.Ln;
END proc1
```

Die Programme ausführt. :

- 1 2

Literaturen :

1) N. Wirth and M. Reiser. Programming in Oberon - Steps Beyond Pascal and Modula. Addison-Wesley, 1992

2) N. Wirth. The programming language Oberon. Software Practice and Experience, 18(7):671–690, July 1988

3) K. Jensen and N. Wirth. PASCAL - User Manual and Report, volume 18 of Lecture Notes in Computer Science. Springer, 1974

4) ETH Oberon Home Page: http://www.oberon.ethz.ch

5) Oberon-Homepage der Landesfachkommission Informatik in Th¨uringen:
http://www.th.schule.de/th/lfk-informatik/oberon/download