Differences between Modula-2 and Pascal

Hausi A. Muller

Department of Computer Science
Rice University
P.O. Box 1892
Houston, Texas 77251

1. Introduction

This paper outlines syntactical and semantical differences between Modula-2 and Pascal. Readers who know Pascal and intend to learn Modula-2 will benefit most from this document.

Modula-2 features which have no immediate correspondent in Pascal are not discussed in this paper. Such novelties include modules, local modules, separate compilation, definition modules, implementation modules, import/export lists, opaque types and multiprogramming facilities.

For a programmer's tutorial and a clear, concise report of Modula-2 refer to [Wirth 83].

2. Syntactic sugar

<table>
<thead>
<tr>
<th>Feature</th>
<th>Modula-2</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>case of reserved words</td>
<td>upper case; WHILE</td>
<td>case is not significant; while = WHILE</td>
</tr>
<tr>
<td>case of identifiers</td>
<td>case is significant; isEmpty</td>
<td>case is not significant; isEmpty = isEmpty</td>
</tr>
<tr>
<td>number of significant characters in identifiers</td>
<td>all characters are significant</td>
<td>a fixed number of characters is significant</td>
</tr>
<tr>
<td>character constants</td>
<td>'@', '&quot;&quot;</td>
<td>'@'</td>
</tr>
<tr>
<td>strings</td>
<td>&quot;That's incredible!&quot; 'Codeword &quot;Barbarossa&quot;'</td>
<td>'That''s incredible!' 'Codeword &quot;Barbarossa&quot;'</td>
</tr>
<tr>
<td>single and double quotes in one string</td>
<td>cannot be done</td>
<td>'&quot;That''s incredible!&quot;'</td>
</tr>
<tr>
<td>comments</td>
<td>(* *); may be nested</td>
<td>{ } (* * *); must not be nested</td>
</tr>
<tr>
<td>set brackets</td>
<td>{}</td>
<td>[]</td>
</tr>
<tr>
<td>constant NIL</td>
<td>standard identifier</td>
<td>reserved word</td>
</tr>
</tbody>
</table>
function keyword

PROCEDURE

FUNCTION

parameterless function declaration

PROCEDURE p(): CHAR ;

FUNCTION p: CHAR ;

sequence of declarations in a block; the symbols [], {}, | are meta-symbols of EBNF (Extended Backus-Naur Form, p. 10, [Wirth 83]); angular brackets [] denote optionality of the enclosed sentential form; curly brackets {} denote its repetition (possibly zero times).

{ ConstDec | TypeDec | VarDec | ProcDec }

CONST
N = 32 ;

TYPE
index = [0..N-1] ;

buf =
ARRAY index OF CHAR ;

VAR b: buf ;

PROCEDURE BufHandler ;
END BufHandler ;

TYPE
entry =
RECORD
  a: CHAR ;
  b: CARDINAL
END ;

sequence =
ARRAY [0..2*N-1] OF entry ;

PROCEDURE BufHandler ;
END ; (* BufHandler * )

storage allocation

FROM Storage IMPORT
ALLOCATE ;
NEW(x)

storage deallocation

FROM Storage IMPORT
DEALLOCATE ;
DISPOSE(x)

separator in variant records and case statements

PROCEDURE BufHandler ;
END BufHandler ;

PROCEDURE BufHandler ;
END ; (* BufHandler * )

repeating procedure identifier
indirect recursion
PROCEDURE a;
BEGIN
  b
END a;
PROCEDURE b;
BEGIN
  a
END b;

PROCEDURE b; FORWARD:
PROCEDURE a;
BEGIN
  b
END; (* a *)
PROCEDURE b;
BEGIN
  a
END; (* b *)

3. Types

<table>
<thead>
<tr>
<th>Feature</th>
<th>Modula-2</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>subrange type</td>
<td>index = [0..31]</td>
<td>index = 0..31</td>
</tr>
<tr>
<td>unsigned integers</td>
<td>standard type CARDINAL</td>
<td>cardinal = 0..MaxCard</td>
</tr>
<tr>
<td>pointer type</td>
<td>(x = \text{POINTER TO } t)</td>
<td>(x = ^t)</td>
</tr>
<tr>
<td>bit access type</td>
<td>standard type BITSET</td>
<td>PACKED SET OF 0..WordLengthMinusOne</td>
</tr>
<tr>
<td>procedure type</td>
<td>a procedure can be an object like a variable of any other type</td>
<td>procedure parameters only</td>
</tr>
<tr>
<td>parameterless procedure type</td>
<td>PROC</td>
<td>not defined</td>
</tr>
<tr>
<td>variant records</td>
<td>no restrictions; same syntax as case statements; RECORD CASE BOOLEAN OF TRUE: u,v: INTEGER</td>
<td>explicit variant must be the last record entry; RECORD CASE BOOLEAN OF TRUE: (u,v: INTEGER) ;\n</td>
</tr>
<tr>
<td>array declaration</td>
<td>ARRAY [1..3], ['a'..'z'] OF INTEGER</td>
<td>ARRAY [1..3, 'a'..'z'] OF INTEGER</td>
</tr>
<tr>
<td>string type</td>
<td>ARRAY [1..N] OF CHAR</td>
<td>PACKED ARRAY [1..N] OF CHAR</td>
</tr>
<tr>
<td>packing</td>
<td>not defined</td>
<td>PACKED StructuredType</td>
</tr>
</tbody>
</table>
unbounded array parameters, arrays of varying length

open arrays;

PROCEDURE p(a: ARRAY OF CHAR)

implementation dependent

low-level storage unit type

WORD; to be imported from module SYSTEM;
compatible with CARDINAL, INTEGER, BITSET, pointers

variant records

address manipulation type

ADDRESS; to be imported from module SYSTEM;
compatible with CARDINAL;
cardinal arithmetic

variant records

address =

RECORD

CASE BOOLEAN OF

TRUE: (p: pointer);
FALSE: (a: INTEGER)

END

4. Statements

Feature Modula-2 Pascal

statement terminator each statement has an explicit terminating symbol; UNTIL for the RepeatStatement and END the rest; no compound statement

statement or SimpleStatement

BITSET assignment x := \{3\}
x := [3]

arbitrary set assignment y := TypeId\{A,B\}; TypeId is CHAR, INTEGER, CARDINAL, enumeration, subrange type ident

y := [A,B]

ReturnStatement PROCEDURE p(): CHAR;
BEGIN
StatementList;
IF b THEN
RETURN('@')
END;
StatementList;
RETURN('!')
END p;

ReturnStatements can also appear in procedures and modules without expression.

FUNCTION p: CHAR;
LABEL 1:
BEGIN
StatementList;
IF b THEN
BEGIN
p := '@';
GOTO 1
END;
StatementList;
p := '!';
1:
END; (* p *)
IfStatement

IF b1 THEN
  a := 3
ELSIF b2 THEN
  a := 4
ELSE
  a := 5 ;  c := 7
END

CaseStatement

CASE i OF
  2:  StatementList1 |
  3..5:  StatementList2 |
  2*3:  StatementList3
ELSE  StatementList4
END

ForStatement

FOR i := 1 TO 3 DO
  StatementList
END

FOR i := 9 DOWNTO 1 DO BEGIN
  StatementList
END

LoopStatement

LOOP
  StatementList
END

ExitStatement

LOOP
  StatementList1 ;
  IF b THEN EXIT END ;
  StatementList2
END

GotoStatement

not defined

IF b1 THEN a := 3 ELSE
BEGIN
  IF b2 THEN a := 4 ELSE
  BEGIN
    a := 5 ;  c := 7
  END
END

IF (i>0) AND (i<5) THEN BEGIN
  CASE i OF
    2: BEGIN StatementList1 END :
    3: BEGIN StatementList2 END :
    2*3: BEGIN StatementList3 END :
    ELSE  StatementList4
  END (* case *)
END ELSE BEGIN
  StatementList4
END (* if * )

FOR i := 1 TO 3 DO BEGIN
  StatementList
END

FOR i := 9 DOWNTO 1 DO BEGIN
  StatementList
END

i := 0 ;
WHILE i <= 55 DO BEGIN
  StatementList ;
  i := i + 5
END

WHILE TRUE DO BEGIN
  StatementList
END

WHILE TRUE DO BEGIN
  StatementList1 ;
  IF b THEN GOTO 1 ;
  StatementList2
END ;
1:

LABEL 1 ;
GOTO 1 ;
1:
5. Expressions and standard procedures

<table>
<thead>
<tr>
<th>Feature</th>
<th>Modula-2</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Evaluation of expressions</td>
<td>&quot;short-circuit&quot;; the AND and OR operators skip the second operand if the expression value can be detected from the first operand.</td>
<td>IF p &lt;&gt; nil THEN BEGIN</td>
</tr>
<tr>
<td></td>
<td>IF (p&lt;&gt;NIL) AND (p^.key&lt;&gt;x) THEN</td>
<td>IF p^ key &lt;&gt; x THEN BEGIN</td>
</tr>
<tr>
<td></td>
<td>StatementList END</td>
<td>StatementList END</td>
</tr>
<tr>
<td>Constant declarations</td>
<td>Constant expressions</td>
<td>Not defined</td>
</tr>
<tr>
<td></td>
<td>N = 100 ; limit = 2*N-1 ;</td>
<td>N = 100 ; limit = 199 ;</td>
</tr>
<tr>
<td>Case labels</td>
<td>Constant expressions</td>
<td>Not defined</td>
</tr>
<tr>
<td></td>
<td>No evaluation order can be assumed</td>
<td></td>
</tr>
<tr>
<td>Scale factor</td>
<td>3.0E+12</td>
<td>3.0E+12 ; 3.0e+12</td>
</tr>
<tr>
<td>Fast increments and decrements</td>
<td>INC(i)</td>
<td>i := i + 1</td>
</tr>
<tr>
<td></td>
<td>INC(i,d)</td>
<td>i := i + d</td>
</tr>
<tr>
<td></td>
<td>DEC(i)</td>
<td>i := i - 1</td>
</tr>
<tr>
<td></td>
<td>DEC(i,d)</td>
<td>i := i - d</td>
</tr>
<tr>
<td>Predecessor</td>
<td>DEC(i)</td>
<td>Pred(i)</td>
</tr>
<tr>
<td>Successor</td>
<td>INC(i)</td>
<td>Succ(i)</td>
</tr>
<tr>
<td>Inverse of ORD</td>
<td>VAL(Typed,ORD(x)) = x; Typeld is CHAR, INTEGER, CARDINAL, enumeration, subrange type ident</td>
<td>Not defined, except for CHAR, CHR(x)</td>
</tr>
<tr>
<td>Round to cardinal</td>
<td>TRUNC(x+0.5)</td>
<td>Round(x)</td>
</tr>
<tr>
<td>Arithmetic functions</td>
<td>Library module</td>
<td>Standard functions</td>
</tr>
<tr>
<td>Low index bound of unbounded array</td>
<td>Always equal to 0</td>
<td>Implementation dependent</td>
</tr>
<tr>
<td>High index bound of unbounded array</td>
<td>HIGH(a)</td>
<td>Implementation dependent</td>
</tr>
</tbody>
</table>
terminate program execution

HALT

LABEL 99 ;
BEGIN
... GOTO 99 ; ...
99:
END.

capitalize character

CAP(ch)

IF ch IN ['a'..'z'] THEN
ch := CHR(ORD(ch) - ORD('a') + ORD('A'))

symmetric set difference

A / B

(A-B) + (B-A)

set inclusion

INCL(S,i)

S := S + [i]

set exclusion

EXCL(S,i)

S := S - [i]

not equal

<>

<<>

logical and

AND &

AND

address of a variable x

ADR(x); to be imported from module SYSTEM

not defined

number of storage units assigned to variable x

SIZE(x); to be imported from module SYSTEM

not defined

number of storage units assigned to variable of type t

TSIZE(t); to be imported from module SYSTEM

not defined

cardinal to real conversion

FLOAT(x)

implicit conversion

type transfer functions; representation is not changed

variables of type WORD, CARDINAL, INTEGER, REAL, BITSET, ADDRESS, and pointers can be transferred into each other; x := Typeld(y); Typeld is one of the above; x is a variable of this type; y is a variable of one of the above types

using variant records
6. Miscellaneous facilities

<table>
<thead>
<tr>
<th>Feature</th>
<th>Modula-2</th>
<th>Pascal</th>
</tr>
</thead>
<tbody>
<tr>
<td>input and output facilities</td>
<td>not defined in the language; library modules;</td>
<td>defined in the language</td>
</tr>
<tr>
<td>compilation units</td>
<td>main MODULE</td>
<td>implementation dependent</td>
</tr>
<tr>
<td></td>
<td>DEFINITION MODULE</td>
<td></td>
</tr>
<tr>
<td></td>
<td>IMPLEMENTATION MODULE</td>
<td></td>
</tr>
<tr>
<td>static variables</td>
<td>variables at a module level</td>
<td>one set of global variables only</td>
</tr>
<tr>
<td>dynamic storage allocation scheme</td>
<td>can be explicitly programmed</td>
<td>implicit; cannot be altered</td>
</tr>
</tbody>
</table>

References


