Modula-2 vs. Pascal

- Unlike Pascal, Modula-2 is case sensitive. All keywords are in CAPITALS.
- Unlike Pascal, all control structures have matching ENDS.
- Comments can be nested, like this: (* (* Hi! *) Bye! *)
- There are no gotos.
- There’s no automatic conversion between integers and reals.
- Boolean expressions are short-circuit.

Control Structures

IF boolean expression THEN
  statement-sequence
ELSIF boolean expression THEN
  statement-sequence
ELSIF boolean expression THEN
  statement-sequence
ELSE
  statement-sequence
END

WHILE boolean expression DO
  statement-sequence
END

History

- Modula-2 is a descendant of Pascal, also designed by Niklaus Wirth. There was an intermediate language called “Modula” but it never caught on. Modula-3 was not designed by Wirth but by a committee from Olivetti and DEC.
- Modula-2 also traces its ancestry from Mesa, a language designed at Xerox. The joke is that Modula-2 is what Wirth remembered of Mesa after he returned from a sabbatical at Xerox, slightly drunk and jet-lagged from the trans-atlantic flight.
Declarations

- Modula-2 has a less strict declaration order than Pascal. Declarations can be given in any order as long as names are declared before use. The exception is procedures which can be declared in any order.
- There is a new equivalence type introduced which gives a new name to a type.
- Records can have arbitrarily many variant parts (CASE).
- There is no special FUNCTION keyword.
- There is a special function type.

```
TYPE equivalence = type;
TYPE subrange = [from..to];
TYPE enumeration = (id,id,...);
TYPE array = ARRAY range OF type;
TYPE record = RECORD
    field : type;
    field : type;
    CASE tag: type OF
        ...;
    END;
    CASE tag : type OF
        ...;
    END;
END;
```

Control Structures...

```
LOOP
    statement-sequence (* EXIT can occur here. *)
END

designator := expression;

REPEAT
    statement-sequence
UNTIL boolean expression

(* The BY-part is optional. step must be a constant.*)
FOR i := from TO to [BY step] DO
    statement-sequence
END

CASE expression OF
    case list: statement-sequence |
    case list: statement-sequence |
    case list: statement-sequence |
    ELSE statement-sequence END

(* To return a value from a function: *)
RETURN expression;
```
Open Array Parameters...

PROCEDURE P (w : ARRAY OF INTEGER);
BEGIN
FOR i := 0 HIGH(w)-1 DO
  ...w[i]...
END P;

VAR x : ARRAY [1..10] OF INTEGER;
VAR y : ARRAY [1..100] OF INTEGER;
BEGIN
  P(x);
  P(y);
END

Types

- CARDINAL is an unsigned integer type, which Pascal doesn’t have.
- BITSET is a special set the exact size of a machine word.
- SYSTEM.WORD is a type the exact size of a machine word which is compatible with all other types of this size.

Open Array Parameters

- Unlike Pascal, Modula-2 has open array parameters which allows you to pass arrays of any size to a procedure:

  PROCEDURE P (w : ARRAY OF INTEGER);

- An open array of SYSTEM.WORD matches any argument:

  PROCEDURE P (w : ARRAY OF WORD); ...
Types...

- You can do arbitrary type conversions, as long as the types are the same size. Conversions convert static types only, no bits actually change.

```pascal
VAR x : ARRAY [1..1] OF INTEGER;
VAR y : INTEGER;
VAR z : CARDINAL;
BEGIN
  z := (CARDINAL)x;
  y := (INTEGER)z + 6;
END
```

Separate Compilation...

- Eventually it was realized that a more formal approach had to be taken to the definition of separately compiled modules. A number of languages (Mesa, Modula-2, Ada, ...) constructed module systems built on the ideas of David Parnas:

The specification must provide
1. to the intended user all the information that he will need to use the program, and nothing more.
2. to the implementer all the information about the intended use that he needs to complete the program, and no additional information.

Separate Compilation...

- Each module has two parts, the specification and the implementation. Much like `.h` and `.c` files in C, only each part is separately compiled.

Separate Compilation...

- From the very beginning of language design history, it was realized that monolithic languages (the entire program is stored in one file and compiled all at once) were no good.
- Monolithic languages made compilation slow and made it difficult for several programmers to work on the same problem.
- As early as 1958, FORTRAN II had separately compiled procedures!
IMPLEMENTATION MODULE IntStack;
  TYPE Stack = POINTER TO RECORD
    space : ARRAY [1..100] OF INTEGER;
    index : CARDINAL;
  END;
  PROCEDURE Create () : Stack;
  BEGIN ... END Create;
  PROCEDURE Destroy (VAR S : Stack);
  BEGIN ... END Destroy;
  PROCEDURE Push (S : Stack; E : INTEGER);
  BEGIN ... END Push;
  PROCEDURE Pop (S : Stack; VAR E : INTEGER);
  BEGIN ... END Pop;
END IntStack.

Using a Module

MODULE Main;
  IMPORT IntStack, Storage;
  VAR S : IntStack.Stack;
BEGIN
  S := IntStack.Create ();
  IntStack.Push (S, 314);
  IntStack.Destroy (S);
END Main.

Definition Module

DEFINITION MODULE IntStack;
  TYPE Stack;
  PROCEDURE Create () : Stack;
  PROCEDURE Destroy (VAR S : Stack);
  PROCEDURE Push (S : Stack; E : INTEGER);
  PROCEDURE Pop (S : Stack; VAR E : INTEGER);
END IntStack.

Modula-2 Modules

- The information that the stack uses an array implementation is hidden within the module’s implementation unit, which is available only to the module’s implementer.
- Note that the Stack type is implemented as a pointer. This is in contrast to an Ada implementation which used a static representation.
- Note that – since Modula-2 does not support garbage collection – we need explicit procedures for memory allocation and deallocation.
Generic Modules

- Modula-2 does not support Generic modules, but much like in C, this can be simulated using untyped pointers.
- `SYSTEM.ADDRESS` is equivalent to C’s `void*`.

Using a Generic Module

```
MODULE Main;
IMPORT GenStack, Storage;
VAR S : GenStack.Stack;
   E : POINTER TO INTEGER;
BEGIN
   S := GenStack.Create ();
   NEW (E); E^ := 314;
   GenStack.Push (S, E);
   GenStack.Destroy (S);
END Main.
```

Generic Definition Module

```
DEFINITION MODULE GenStack;
IMPORT SYSTEM;

TYPE Stack = POINTER TO RECORD
   space : ARRAY [1..100] OF SYSTEM.ADDRESS;
   index : CARDINAL;
END;
PROCEDURE Create () : Stack;
BEGIN ... END Create;
PROCEDURE Destroy (VAR S : Stack);
BEGIN ... END Destroy;
PROCEDURE Push (S : Stack; E : SYSTEM.ADDRESS);
BEGIN ... END Push;
PROCEDURE Pop (S:Stack; VAR E:SYSTEM.ADDRESS);
BEGIN ... END Pop;
END GenStack.
```
Modula-2 has a special SYSTEM module that contains any system-specific definitions.

**Dynamic Allocation**

- There is a special module `Storage` that exports two procedures `ALLOCATE` and `DEALLOCATE`.
- The two built-in procedures `NEW` and `DISPOSE` are translated into calls to `ALLOCATE` and `DEALLOCATE`.
- This allows us, at least in theory, to write our own storage allocators.
- Modula-2 does not support garbage collection.

**IO**

- Modula-2 has no defined IO procedures like Pascal’s `read` and `write`.
- Instead, each implementation was supposed to define its own set of standard modules to do IO and related systems functions.
- There were some standard modules for IO, but even them weren’t always compatible.
- This made porting a Modula-2 program much harder than it should have been.

Modula-2 also has local modules.
- Local modules can be nested within each other, within procedures, etc.
- No sane person ever used them.
Facilities for Systems Programming

- The `SYSTEM` module also has functions for constructing co-routines (`NEWPROCESS` and `TRANSFER`).
- The `SYSTEM` module also has a function `IOTRANSFER` for handling interrupts. Modules can be given an interrupt priority:

  ```
  MODULE Printer[2];
  ...
  BEGIN
  ...
  END Printer;
  ```

Readings and References

- [http://murray.newcastle.edu.au/users/staff/peter/m2/Modula2.html](http://murray.newcastle.edu.au/users/staff/peter/m2/Modula2.html)
- [http://www.modulaware.com/m2wr](http://www.modulaware.com/m2wr)
- [http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/web/GNUModula2.html](http://floppsie.comp.glam.ac.uk/Glamorgan/gaius/web/GNUModula2.html)
- A Modula-2-to-C translator is available on [lectura](http://home/cs520/2003/bin/m2c). It can also be downloaded from here: [http://www.mathematik.uni-ulm.de/modula](http://www.mathematik.uni-ulm.de/modula)