### Variable Declarations

```pascal
var
list of identifiers : data type;
list of identifiers : data type;
list of identifiers : data type;
```

- Builtin datatypes: integer, real, char, boolean

### Arithmetic

- integer and real
- `+,-,*,/`,rem, mod.
- Integers are promoted to reals when necessary.
- Builtin functions: `abs`, `arctan`, `cos`, `exp`, `ln`, `round`, `sin`, `sqr`, `sqrt`, `trunc`.
- Builtin functions (integer, char, boolean only): `chr`, `ord`, `pred`, `succ`.

### Structure of a Pascal Program

```pascal
PROGRAM Name (list of files);
CONST
(* Constant declarations *)
TYPE
(* Type declarations *)
VAR
(* Variable declarations *)
(* Procedure and function definitions *)
BEGIN
(* main program *)
END.
```

- Note the strict ordering of declarations
**Statements: if**

if boolean expression then
  statement
else
  if boolean expression then
    statement
  else
    begin
      statement
      statement
      statement
    end

- The `else` is always matched with the closest nested `if`.

**Statements: case**

case ordinal expression of
  list of cases: statement;
  list of cases: statement;
  list of cases: statement;
  otherwise statement
end;

- `otherwise` is optional.
- The `list of cases` looks like this: 1,2,7..9. I.e. it can contain ranges.

**IO**

- `read(X,Y,Z)` reads the next three values from an input file into the variables `X,Y,Z`. Spaces are skipped.
- `readln(X,Y,Z)` is the same as `read(X,Y,Z)` except the end-of-line is also eaten.
- `write(X,Y,Z)` write the values of the variables `X,Y,Z`.
- `writeln(X,Y,Z)` is the same as `write(X,Y,Z)` except and end-of-line is also written.
- There is some limited formatting possible. `write(X:10, 9:4, 45:2);` uses the `:fieldwidth` syntax.

**Boolean Expressions**

- `<,>,=,<,>=,<>` compares integers, reals, chars, booleans.
- `not, and, or, xor` combine boolean expressions.
- Pascal does not use short-circuit evaluation. Hence, this makes for problems:
  
  ```
  if (x<>0) and (y/x > 5) then
  ```

- Pascal has non-intuitive precedence:
  
  ```
  4 > 8 or 11 < 3
  ```

  is parsed as
  
  ```
  4 > (8 or 11) < 3
  ```

  Hence, it becomes necessary to insert parenthesis.
**Statements: goto**

- Pascal has no exception handling mechanism. Gotos were the only way of, say, jumping to the end of the program on an unrecoverable error.
- Labels have to be integers and have to be declared.

```pascal
procedure P ();
  label 999;
  goto label;
  ... goto 999;
  999:
  label: end;
```

**Procedures**

PROCEDURE Name (list of formals);
  CONST (* Constant declarations *)
  TYPE (* Type declarations *)
  VAR (* Variable declarations *)
  (* Procedure and function definitions *)
BEGIN
  (* procedure body *)
END;

- Note the similarity with the program structure.
- Note that procedures can be nested.
- Note the semicolon after the end.

**Statements: for**

for index := start to stop do
  statement;
for index := start downto stop do
  statement;

- The index must be declared outside the loop.
- Only ordinal datatypes are allowed.

**Statements: loops**

while boolean expression do
  statement;
repeat
  statement;
  statement;
  until boolean expression;

- Note the asymmetry: the while statement body can only contain one statement.
Procedures...

- Procedures can be nested:
  
  ```pascal
  procedure A ();
  procedure B ();
  begin
    ...
  end;
  begin
    ...
  end;
  ```

- Names declared in an outer procedure are visible to nested procedures unless the name is redeclared.

- Procedures can be recursive. The `forward` declaration is used to handle mutually recursive procedures:

  ```pascal
  procedure foo (); forward;
  procedure bar ();
  begin
    foo();
  end;
  procedure foo();
  begin
    bar();
  end;
  ```

Procedures...

- Formal parameters look like this:

  ```pascal
  procedure name (formal1:type1; formal2:type2; ...);
  or like this
  procedure name (formal1,formal2...:type1; ...);
  ```

- By default, arguments are passed by value. `var` indicates that they are passed by reference:

  ```pascal
  procedure name (var formal1:type1; ...);
  ```

Procedures...

- Functions are similar to procedures but return values:

  ```pascal
  function func1 (formals);
  begin
    func1 := 99;
  end;
  ```

- To return a value assign it to the function name.
Types: Sets

\[
\text{type } S = \text{set of } \text{type};
\]

\[
\text{type letter } = 'A' .. 'z';
\]

\[
\text{type letset } = \text{set of letter}; (* \text{Only ordinal types}.*)
\]

\[
\begin{array}{c}
\text{var } x, y, z, w: \text{letset}; \\
\text{begin} \\
\quad x := ['A'..'Z','a']; y := ['a'..'z']]; \\
\quad z := x + y; (* \text{set union} *) \\
\quad z := x * y; (* \text{set intersection} *) \\
\quad w := x - y; (* \text{set difference} *) \\
\quad \text{if} 'A' \text{in } z \text{then} ...; (* \text{set membership} *) \\
\text{end.}
\end{array}
\]

Types: Enumerations

- Enumerated types are declared like this:

\[
\text{type enum } = (id1, id2, ...);
\]

\[
\text{type month } = (\text{Jan, Feb, Mar, ...});
\]

\[
\begin{array}{c}
\text{var } x: \text{month}; \\
\text{begin} \\
\quad x := \text{Jan}; \\
\quad \text{write}(\text{ord(Jan)}); (* \text{Prints 1} *) \\
\text{end.}
\end{array}
\]

Types: Arrays

\[
\text{type arr } = \text{array [ enumeration type ] of element type};
\]

\[
\text{type letter } = 'A' .. 'z';
\]

\[
\text{type A1 } = \text{array [letter] of integer};
\]

\[
\text{type A2 } = \text{array [1..4,9..16] of boolean};
\]

\[
\text{type A3 } = \text{packed array [0..100] of char};
\]

\[
\begin{array}{c}
\text{var } x: A1; y:A2; z:A3; \\
\text{begin} \\
\quad x['B'] := 4; \\
\quad y[4,10] := \text{true}; \\
\quad y[4,1] := \text{true}; (* \text{Runtime error.} *) \\
\quad z := "Hello World!";
\text{end.}
\end{array}
\]

Types: Subranges

- Subrange types are declared like this:

\[
\text{type range } = \text{low .. high};
\]

\[
\text{type letter } = 'A' .. 'z';
\]

\[
\text{type summer } = \text{Jun .. Aug};
\]

\[
\begin{array}{c}
\text{var } x: \text{summer}; \\
\text{begin} \\
\quad x := \text{Jul}; (* \text{OK} *) \\
\quad x := \text{Dec}; (* \text{Runtime error.} *) \\
\text{end.}
\end{array}
\]
Types: Arrays

```pascal
type arr = array [ enumeration type ] of element type;
type letter = 'A' .. 'z';
type A1 = array [letter] of integer;
type A2 = array [1..4,9..16] of boolean;
type A3 = packed array [0..100] of char;

var x: A1;
y:A2; z:A3;
begin
  x['B'] := 4;
y[4,10] := true;
y[4,1] := true; (* Runtime error. *)
  z := "Hello World!";
end.
```

Types: Variant Records

```pascal
type name = record
  field name : type;
  field name : type;
  case tag name : type of
    case : (field name:type;...);
    case : (field name:type;...);
end;
```

Types: Records

```pascal
type name = record
  field name : type;
  field name : type;
end;
type rec = record a : integer; b : boolean; end;

var r: rec;
begin
  r.tag := true; r.x := 55;
  r.tag := false; r.y := 'A';
  r.tag := true; r.y := 55; (* Runtime error? *)
end.
```
**Types: Variant Records...**

```pascal
type rec = record
  a : integer;
  case boolean of
    true : (x : integer);
    false : (y : char);
  end;

var r: rec;
begin
  r.x := 55; r.y := 'A';
end.
```

- This construct is used to bypass Pascal’s strong typing.

**Types: Pointers**

```pascal
type name = "type; type ptr = "rec;
type rec = record a : integer; b : boolean; end;

var x: ptr;
begin
  new(x);
  x^.a := 55;
  write(x^.b);
end.
```

**Readings and References**

- [http://dmoz.org/Computers/Programming/Languages/Pascal](http://dmoz.org/Computers/Programming/Languages/Pascal)
- [http://www.gnu-pascal.de](http://www.gnu-pascal.de)