A Very Brief Overview of Pascal

- Program layout, example
- Declarations, etc
- Expressions
- Basic statements: if, case, iteration
- I/O
- Sets
- Pointers, etc
- Records
- Bit manipulation

Layout

- Layout

```
program myprogram(input, output);
  const ...
  type ...
  var ...

  function something(...):integer;
    const ...
    type ...
    var ...

    procedure another(...)
      var ...
       begin  { procedure another }
       end;
    begin   { function something }
    end;

begin { main }
end.
```

Layout and Basics

- Comments
  - { ... } and (* ... *)
- Pascal is not case SENSITIVE
- Identifiers: letters & digits, begin with a letter
- Begin/End, not { }
- Built-in types
  - char, integer, real, Boolean
  - C has: char, int, long, short, float, double, long double

Layout and Basics

- Type keyword
  - type Color = (red, green, blue);
  - C has enum Color {red, green, blue}; Const keyword
- const Pi = 3.14;
  Rate = 0.05;
  Greeting = 'Hello there!';
  - C has const and typedef
- Variable declarations: var keyword
  - var r: real;
    i, j: integer;
- Arrays
  - astring: packed array [10..90] of char;
  - grid: array [5..25, 0..10] of integer;
  - C has int grid [16] [11];

Subroutines

- Procedures and Functions
  - procedure and function keywords
- Procedures defined before/above use within visible scope
- Parameters and return values
  - procedure swap (var x, y : integer);

```
...
begin
...
end;
```

  - function swap (var x, y: integer) : boolean;
  - procedure quadEq(a, b, c: real; var x1, x2: real);
  - the *var* keyword indicates pass by reference

Expressions

- Arithmetic expressions
  - := (assignment), +, -, *, / (real division), div (integer division), mod (remainder)
  - no i++, ++i, i--, --i
- Automatic conversion from integer to real
- Built-in functions
  - round(x), trunc(x), ord(ch), chr(n), succ(ch), pred(ch)
  - abs, sqrt, sin, cos, exp, ln, sqr, arctan
- Relational ops
  - =, <>, <, <=, >, >=
- Logical ops
  - and, or, not

Control Statements

- if <condition> then
  <stmt> { NO semicolon ; even with begin/end }
  else
  <stmt>; { use begin/end if you need mult stmts }
- Short-circuit evaluation not standard
- Iteration
  - while <cond> do begin ... end;
  - for i := n1 to n2 do begin ... end;
  - for i := n2 downto n1 do begin ... end;
  - for ch := let1 to let2 do begin ... end;
  - repeat <stmt> ... until <cond>;
- No break or continue statements

More Control & I/O

- case <exp> of
  <const1>: <stmt1>;
  ...
  otherwise: <stmt>; { or default in some dialects }
  end;
  - <const> can be char, int, boolean
  - <const> can be a comma-separated list
- Input/Output
  - write(x, y, ...);
  - writeln(x);
  - writeln;
  - ditto read();
  - writeln('$', amt : 6: 2); { formatted $ 19.99 }

Sets & Bit Manipulation

- **Sets**
  - type <settypename> = set of <sometype>;
  - typically bitmapped with limited max size, 256
- **No bit manipulation**

Pointers

```
type
    RealArr = array [10..99] of real;
var
    i : integer;
    pi1. pi2: ^integer;   { pointers to integers }
    pa : ^Real Arr;       { pointer to a array of reals }

begin
    i := 99;
    new (pi1);
    if pi1 = nil then
        writeln('Mem error');
    pi1^ := i;
    pi2 := pi1;
    writeln (pi2^);
    dispose (pi1);
    new (pa);
    pa^[1] := 1.123;
    pa^[2] := pa^[1];
    writeln(pa^[2]);
    dispose(pa);
end.
```

- **No connection between pointers and arrays (like C has)**

Records

- type
```
PointType = record
                x, y : real;
end;

RectType = record
            upperL, lowerR : PointType;
            color: integer;
end;

var
  rect : RectType;
```

- Access: rect.color := 255;
  - pointer access: newNode^.info

- Varient record like unions in C
```
RECORD
  Part : 1..9999;
  CASE On_Order : Boolean OF
      TRUE  : ( Order_Quantity : INTEGER;
                Price          : REAL );
      FALSE : ( Rec_Quantity   : INTEGER;
                Cost           : REAL );
  END;
```