WHARTON83
Wharton, R.M.; A Note on Types and Prototypes; ACM SIGPLAN Notices 18, 12 (December 1983) pp. 122-125.

In this paper, Wharton discusses the notion of "type" in programming languages and introduces a competing, but simpler notion "prototype". The paper asserts that a language with prototypes has the same (computational) power as one with types yet with fewer rules and a program written in a language with prototypes will have a smaller name space than the corresponding program written in a language with types.

The author begins with a dictionary definition of the two terms and then discusses the concept of a type as used in the language Pascal. He notes that in Pascal the rules for composing types are complex and in Ada they are almost incomprehensible. For example in Pascal you can have sets of scalars and arrays of sets but not sets of arrays. He also points out that in Pascal the programmer has the option of naming the type or using it anonymously i.e.

```
type range = 1 .. 10;
var x : array [ range ] of char;
var y : array [ 1 .. 10 ] of char;
```

and worse still there are some Pascal implementations where there are restrictions as to where anonymous types can be used, i.e. procedure formal parameters must be declared with named types.

After exposing the inconsistencies and complexities of Pascal types, Wharton presents his construct. In the prototype model, the language contains a few primitive variables (rather than types) i.e. char, integer, ... and the programmer declares new instances of variables with declarations of the form:

declare a **like** integer;
declare b **like** a;

That is, the process of declaring a variable can be viewed as "cloning" a variable from a prototype. Subranges and arrays can be declared using similar syntax i.e.

declare bounds = lower .. upper;
declare vector = **array** [lower .. upper] **like** real;

The author closes the paper with a discussion of the advantages of prototypes over types. First, prototypes are a simpler concept since a prototype is simply a variable (primitive or composite) whereas a type is a class of variables. He argues that in Pascal there can be confusion about the differences between types and variables and a programmer might try an perform an operation on a type which is only correct for a variable, etc. He also argues that in Pascal one has the option of naming the type which is only adds unnecessary complexity and serves to promote variation and confusion among programming styles. However, prototypes avoid these problem completely and reduce the program name space since there are no types to name.

This paper serves as an introduction to a novel construct to replace the type concept in Pascal like languages, however the author fails to give a full specification of the mechanism and does not fully address the issue of composite structures. In particular, he does not discuss record structures or the semantics of creating new variable prototypes.