WALLIS80
Wallis,Peter J.L.;
External Representations of Objects of User-Defined Type;
ACM Transactions on Programming Languages and Systems, 2, 2 (April 1980).
This paper considers the problem of representing a value of an abstract data type in human-readable form. This external representation problem arises in program input/output and in writing literals in the program text. The author describes an extension to his programming language PPL to permit description of the external representation of objects of user-defined types.

The ability to define new types is available in many modern programming languages. Typically, the programmer defines the new type by describing its representation in terms of more basic types. This process terminates with the so-called "primitive" types, those that are always part of the langauge and which it is the compiler writer's responsibility to implement.h

The distinction between a programmer-defined type and a pre-defined type varies from language to language, but it is usually the case that no external representation for values of a programmer-defined is defined. This means that the normal text I/O facilities of the language are not available. Furthermore, it is not possible to write literals of the new type in the program text. This creates a distinction between objects with primitive types and objects of user defined type.

The extension to PPL described here attempts to solve this problem. New types are defined in PPL within a "cluster". The extension adds a new "extrep" (external representation) specification to the cluster. The external representation of a type is defined in terms of the external representation of previously defined types and possible extra characters, called "framing characters". This specification is in the form of a "template", which is similar to a Fortran FORMAT, or to the format parameter to "printf()" in UNIX C. Since the template alone is not sufficient to describe the repesentation, two processing sections are provided to describe the necessary transformations for input and for output.

The author then describes how to use the specifications at run-time to do input and output. This uses the format string to control the invocation of the input/output transformation functions, other external representations, and the recognition or output of framing characters. The external representations of various primitive PPL types is also described. The required processing of extreps by the compiler is considered.

Programmer-level I/O commands are briefly discussed. These use a format driven scheme, in which a format string composed of extrep names is followed by an arbitrary collection of parameters that represent objects (input) or values (output).

The handling of literals in the program text is considered next. This is difficult because the description of how to process the literal is also in the program text. The adopted solution is to develop a lexical syntax (using a "quoting" convention) that allows the compiler to determine the character string that makes up the literal without reference to the corresponding extrep. These strings are saved and turned into actual values during a program loading process that invokes the (now-compiled) extrep code.

The paper concludes with a few simple examples and some suggestions for future investigations.

The major difficulty with the proposed scheme is that it is not flexible enough.Input in particular is sharply constrained by the limited power of the format strings. Something at least as powerful as regular expressions is probably needed. On output, the author points out that it is not possible to describe effectively the representation of complex objects that require multiple line formatting.

In the end, the most effective solution is probably similar to the one adopted by Smalltalk-80. Each abstract data type that needs an external representation provides an operation to generate such a representation given a value of the type. It may also provide an operation to interpret a string of characters as a value of the type. It may also provide an operation to interpret a string of characters as a value of the type. This

scheme provides the needed flexibility and requires no special additions to the programming language.