LISKOV00
Liskov, Barbara et al.; Abstraction Mechanisms in CLU; CACM 20, 8 pp. 564-76.

The CLU programming language is primarily designed to support abstraction as a programming methodology. Liskov maintains that CLU implements three kinds of abstraction - procedural, data and control.

Procedural abstraction in CLU is essentially the same as in most Pascal-like languages.

The primary data abstraction mechanism in CLU is the cluster. A cluster is similar to a module in its intent to hide information from the outside environment, however, a cluster goes further as a data abstraction. CLU takes the view that the relevant attributes of a data abstraction are not only the type of the data but also the operations that are meaningful on that type. A cluster attempts to enforce this notion at the compilation level.

A cluster is a set of procedures that operate on the underlying representation. An example taken from Liskov's paper is that of a binary tree of words. In this case the cluster would be a set of procedures that represent meaningful operations on the tree, such as insertion, deletion and traversal. Along with the procedures is an underlying representation, in this case records with pointers to other records. Cluster procedures are accessed as if they performed operations on the tree. An explicit type conversion occurs in the procedure heading of a cluster procedure to convert to the internal representation which is not visible outside. Clusters may accept parametrized types. This allows one to implement a tree of words or integers or any other type with the same cluster implementation.

CLU provides a control abstraction for iteration. In most languages the 'for' statement is restricted to iteration over integers. In CLU the 'for' statement is constructed as follows:

```
FOR declarations IN iterator_invocation DO
   body
END;
```

An iterator invocation is similar to a procedure. It receives arguments in its invocation, and its 'yield' statement returns values which are assigned to the variables declared in the 'for' statement. Thus the 'for' statement can be used to iterate over any data structure such as a tree or string, by providing an iterator to step through the data structure.

CLU supports separate compilation and incremental program development by means of the CLU Library. The library contains a set of description units, one for each abstraction. The description unit contains the interface specifications for the modules that implement that abstraction. A module that uses a particular abstraction can be compiled even if the implementation for that abstraction does not yet exist. A module can refer to an abstraction by any name, and the library maintains a maping list between arbitrarily named abstractions, their interfaces, and the modules that implement them. This binding takes place just before execution.

Liskov's paper is in the form of a tutorial introduction to CLU by means of an extended example. In addition to abstraction mechanisms Liskov also discusses some features of the CLU implementation and semantic considerations. The key feature is the use of descriptors as a naming mechanism in the implementation. These descriptors point to objects, and thus variables refer to descriptors not to the underlying objects. Variables may share objects by having their descriptors point at the same object.

Overall the paper is easy to understand, well organized and provides a good overview of some of the key features of CLU.