KRULL81

Krull F.N.; Experience With ILIAD: A High-Level Process Control Language; CACM 24, 2 (February 1981) pp. 66-72.

ILIAD is a language developed at General Motors for programming industrial process control systems. It was designed to improve programmer productivity by making programs easier to write and maintain, and to be machine independent. To minimize the transition, the language was designed to be similar syntactically to PL/1. At the time of the article, it had been implemented on 3 16-bit computers, and one 8-bit system.

The compiler for the language produces virtual machine instructions which are interpreted at run time by a monitor. The monitor also controls all I/O and task dispatching. This results in a system which is fairly easy to transfer from one machine to another.

The interpretation was found to be effective, and worked well in applications which were not time critical. The system was only slow in the execution of algorithms which were coded in ILIAD. The functions which are carried out by the monitor ran very quickly, since it was coded in assembler. For Time critical applications a facility to link in Fortran subroutines is supplied.

Due to the lack of program development tools on the small systems, the compiler was implemented as a cross compiler. This worked very well, but the author suggests that a debugging system on the host computer would have been very useful.

The language is structured into modules, which can contain either tasks, which are the unit of concurrency, or they can contain procedures for the support of the tasks. The modules can each be compiled separately. This was found to be useful for developing standard modules for use in different applications, and it also eased the load of compiling.

The sequential statements for ILIAD are all fairly standard. It requires that all variable be declared, and only two storage classes are used. The storage classes are PRIVATE (local to modules, automatic) and SHARED (global, static). This resulted in the SHARED type being used more widely than it should have been. One unique data type is TIME. This is used to keep track of actual time, and there are operators to set, read, add and subtract TIME variables.

Some of the important features of the language are described below. The language allows concurrency, through the dynamic activation of tasks. Mutual exclusion is ensured by a LOCK statement which ensures the exclusive access to data. Syncronization is carried out by the AWAIT and DELAY statements which allow tasks to wait for events, and elapsed time.

The run time monitor handles error conditions in one of two ways, if the error is not too severe, then an error code is set and execution continues. The program can then test the status of this code. If the error is severe, or fatal the system can branch to a user defined exception handling routine.

Some statements (notably the I/O statements) allow optional prepositions to make them more readable. The system also has a simple preprocessor which allows file inclusion, and some simple macros. In addition, keywords are not reserved and no implicit type conversion is done.

The paper claims that the language did in fact promote readability and that the programs were easily readable by someone familiar with the application after a short exposure to ILIAD.