LITTLE Newsletter No. 24

D. Shields
March 19, 1973

Proposals for Next Stage of LITTLE Development

We indicate the major areas with which LITTLE development should focus in the near future, and indicate some of the subproblems of interest. Our intent is to indicate the problems seen to be of immediate interest; and to comment on the relevance of these problems. Suggestions about areas not described in this newsletter are invited.

The status of the LITTLE effort at the present time is as follows:

1) A working FORTRAN-written compiler for LITTLE is available.

2) The lexical-scanner phase of the LITTLE compiler has been transcribed into LITTLE and substantially debugged.

3) The LITTLE compiler has been converted to a table-driven form, and the new compiler and code generators are currently being debugged. We expect the availability of a LITTLE-written parser and generators within a month.

4) LITTLE is already supporting a substantial user community, to wit, the SETL-run time library, the development of a BALM system based on LITTLE, and the development of a LITTLE-written operating system supporting the ARTSPEAK language for mini-computers. Though 'bootstrap' compilers are usually meant only for one-time use, namely, the compilation of the compiler itself, LITTLE has already obtained, even in the bootstrap form, a user community which must be supported.

5) a new assembler for LITTLE, which promises substantially improved code generation at the basic block level, has been written, and is just entering the debugging process.

Thus, within two months we may expect that the LITTLE bootstrapping process will be complete; at which point we may contemplate the transfer of LITTLE to other machines; moreover, since LITTLE is essential to the ongoing development of the SETL language, improvements in LITTLE of a 'local' nature (i.e., directed to use at NYU) may be advisable. As a consequence, we here consider those extensions and refinements of LITTLE which are of immediate interest.

The main areas of concern may be separated into several partially overlapping areas:

-- human factors-- improving LITTLE from the user's point of view
-- LITTLE extensions--changes to the language based on experience so far obtained; e.g., the inclusion of 'DO' and 'WHILE' statements in the language.
-- preparation for transportability--preparing LITTLE for use on other machines and operating system. This involves the isolation of dependencies on the NYU operating system, and may involve the creation of support facilities, e.g. a LITTLE written version of the UPDATE library support program.
-- documentation--closing the 'documentation gap' which currently limits access to the LITTLE compiler to those few people who have been substantially involved in the bootstrap process.

HUMAN FACTORS CONSIDERATIONS

At the present time, LITTLE gives marginal support to its users, in that the compiler itself is unwieldy, requires large amounts of core to execute (thus increasing turnaround time for LITTLE users), provides a minimal imput/output package, and requires an extensive (and obscure) piece of job-control language to run LITTLE programs. The main desiderata from the user's point of view are as follows:

a) Provide better coordination of compiler listing between the scanner and the parser phases--much redundant output is currently generated.

b) Provide better input/output facilites, particularly in the production of listable output. Output is currently generated using the 'PUP' package, or the'UTILE' package within the compiler itself; both package were written to 'minimize coding time' while providing a usable input/output function. However, both packages are very system-dependent and should replaced by facilities modelled along the lines suggested by B.Abes in LITTLE Newsletter number 11.

c) Support 'modular' LITTLE programs, by providing facilities to support libraries of LITTLE-compiled modules, and, more importantly, by providing a name-scoping scheme which is less of a disaster-area than the current scheme (whose only advantage is its ease of implementation). The consequent improvements to LITTLE are:

    1) Redefine listing conventions, based on experience of SETL-library group.
    2) Provide better input/output by implementing (at least part) of Bob Abes's definition of LITTLE input/output
    3) Support modularity by converting to MACE loader, and by implementing 'name-set' conventions proposed by Dave Shields in LITTLE Newsletter number 23, and modify LITTLE to compile constants and temporaries 'locally' within routines, and not globally, as at present.

LITTLE EXTENSIONS.

LITTLE was purposely defined incompletely, so as to defer a precise definition of LITTLE until some user experience had been gained. This experience suggests the following points of interests:

a) Add 'IF-THEN-ELSE', 'WHILE' and 'DO' to the language, instead of supporting these statement forms by macros. This will permit the generation of diagnostic messages which the macros forbids, e.g., check that do-loops are correctly terminated.

b) Clear up ambiguities in language definition, e.g., multi-word field assignments, influence and handling of 'sign' bit, etc., by providing a more precise definition of the basic operations supported by LITTLE.

c) Clarify handling of character-strings , even if a new data-type must be defined. LITTLE admits a marginal representation of strings, which is very machine-dependent.

d) Provide better input/output facilities, as described in the 'human factors' section above.

e) Improve LITTLE optimisation facilities.

f) Implement some debugging features, e.g., check of validity of subscript values in array references. Some form of the 'ASSERT' statement used in the SETLB debugging package may be particularly useful.

TRANSPORTABILITY ISSUES

The LITTLE bootstrap process should be substantially complete soon. Accordingly, we should consider those problems of interest in transporting LITTLE to other machines. The main problems are as follows:

a) Improved documentation within the LITTLE system itself, so as to make the LITTLE system 'self-defining'. Moreover, code fragments resulting from properties of the NYU operating system should be clarified and isolated. Thus substantial effort is needed to document the LITTLE compiler in a machine independent manner, so that the compiler is maximally self defining .

b) Provision of library support facilities similar to those features supplied by the UPDATE program used to maintain LITTLE at CIMS.

c) Clarification and definition of the input/output facilities needed by the LITTLE compiler.

d) A machine-independent definition (as much as possible) of the loader support function required by the LITTLE system.

e) Moreover, we should anticipate the problems to be encountered during the 'transportation' process; and design procedures to minimize the effort of CIMS-based people to support this process. For example, test decks which verify correctness of parts of the LITTLE system should be constructed, and, more importantly, these decks should verify that the new system satisfies the minimal assumptions made when LITTLE movable-system was generated. e.g., that at least 80 characters per line may be printed.

DOCUMENTATION ISSUES

As has been mentioned before, the LITTLE compiler at present suffers from a 'documantation gap', in that only those people who have been involved in the LITTLE bootstrap process have a detailed knowledge of the LITTLE compiler. A primary consideration must be the inclusion within the compiler itself of sufficient documentation to describe both the 'machine-independent' and the'machine-dependent' components of the LITTLE compiler itself. Immediate documentation goals proposed to address these problems are:

1) Provision of a 'global' variable dictionary defining all macros, variables, and codes used in more than one routine.
2) Inclusion of comments of a subroutine nature describing the function, of and global assumptions made concerning each routine in the LITTLE compiler.
3) Definition of all 'global' macros at the start of the program text.
4) Description of assumed system functions - for example, inclusion within the assembler of a description of the structure of the loader-tables which must be produced by the LITTLE compiler.