TO:	DISTRIBUTION	CT	Clingen
FROM:	R. A. FREIBURGHOUSE	FJ	Corbató
SUBJECT:	VERSION II OF THE PL/I COMPILER	RC	Daley
DATE:	AUGUST 21, 1970	JW	Gintell
		JM	Grochow
		JH	Saltzer✓
		TH	VanVleck
		BL	Wolman

The PL/I Compiler rebuilding project has progressed to the point where we now must define those language changes and features which the new compiler (Version II) will support.

The principal design objective of the Version II compiler is to reduce the resource requirements of the compiler while continuing to support the full PL/I language and continuing to generate the same quality object code. It is too early to predict the performance of Version II but it is expected to be significantly faster than the current compiler.

While rebuilding the compiler we expect to eliminate as many implementation restrictions as possible. Several improved error messages will be produced and somewhat better object code will be generated. The two most significant object code improvements will be:

1. Subscripted label constants will become elements of a transfer vector.

2. In many cases, internal procedures and begin blocks will share their parents stack frame and will use a fast call, save, and return.

Nearly all source programs compilable by the current compiler will be compiled properly by Version II. However, those features of the current language which are not included in the ANSI standard will receive warning diagnostics.

We feel that the advantages of coding in a standard language outweigh the disadvantages of having to make a few minor syntactic changes to get rid of warning diagnostics.

The source language compiled by Version II is a subset of the proposed ECMA/ANSI standard PL/I. This language is a cleaner and more powerful language than our current PL/I. In nearly all cases the changes have been favorable from the Multics user and system programmers point of view. The remainder of this document describes in some detail the new features of the Version II compiler language.

Extensions to the current language:

1. Entry variables will be supported as a general data type. They may be dimensioned, members of structures, belong to any storage class, etc.

2. Functions will be able to return any scalar data type except area. Future versions of the compiler will support aggregate valued functions as part of an extension which will include aggregate expressions and assignment.

3. Functions may return string values whose length is determined by the function and not by its caller. Returns (char(*))

4. Based varying strings will be allowed.

5. Null argument lists are allowed in: function references, call statements, procedure statements, entry statements, return attributes and entry attributes.

6. The arithmetic will be extended to include: complex, scaled fixed-point, true decimal fixed-point, and all associated built-in functions.

7. The list of built-in functions will be extended to include: clock, translate, and verify. Several unimplemented functions will be implemented.

8. A new procedure option (   ) which allows programmers to optimize calls will be available. It has the following effect on all calls made by the procedure.

   a. Constant arguments are not copied into temporary storage.

   b. Aligned level one scalar arguments are considered to match unaligned scalar parameters.

Features of ANSI PL/I not supported by Version II:

*1. Aggregate expressions and array cross sections.

*2. Tasking

3. Controlled storage

4. The like, picture, and generic attributes

5. Area assignment

6. True decimal floating-point

*This feature is being extensively redefined by ECMA/ANSI standardization groups.

Changes to the current language:

1. Labels on Declare statements are recognized as labels and effectively become labels of a null statement which replaces the declare statement in the object program. Version I ignores labels on Declare statements.

2. The syntax of a label is restricted to <identifier>[(<decimal-integer>)]: All labels are constants and refer to statements in the current invocation of a block. Subscripted labels are compiled into a transfer vector and thus provide a very efficient implementation of a switch. Version I allowed automatic label variables to appear as labels - a feature which provided a rather inefficient switch. The new syntax is a subset of the ANSI standard - the old form is no longer standard and is not allowed by Version II.

3. The declaration of an entry constant is restricted to the form:

```
     dcl <identifier> entry ([<parameter-desc>])

       [returns ([<returns-desc>])]

       [reducible|irreducible] [external]
```

   This syntax is the proposed ANSI standard and has the following implications:

   The only entry constants (entry names) declared in a procedure are those entry names which are external names of other procedures.

   A procedure statement or entry statement is considered a declaration of the attached label as an entry name with an entry (   ) and returns(   ) attribute. In other words the compiler examines the text of a procedure and builds a declaration for each of its entries and each of its contained internal procedures. This eliminates much redundant declarative information now required from the programmer and lessens the possibility for mistakes.

   All undeclared names used with an argument list or in a call statement are considered built-in names. If the name is one of the built-in functions recognized by this compiler the built-in function will be invoked. Otherwise a Multics call with descripters will be made and the name will be declared as a built-in external entry. Note that standard PL/I would consider this last case an error. We allow it because of non-standard system procedures like ioa_ which cannot legally be declared in standard PL/I. Effectively the set of built-in functions includes everything in the system reachable by the search rule.

3. (Continued)

Built-in functions are no longer declared by default unless they have an argument list. Thus the argumentless built-ins like null, clock, time, date, onchar, and onsource must be explicitly declared built-in.

These changes to the standard language were made to:

1. Reduce the number of source program errors resulting from undeclared entry names. (Argument parameter mis-matches, etc.).

2. To allow the list of built-in functions to be extended by each implementation, and to allow implementation defined built-in or system supplied subroutines.

3. To clean-up the current rules regarding contextual declarations of entry and built-in names.

4. The ANSI standard has been modified so that it now agrees with our implementation of string temporaries.

You will recall that in our implementation a string expression is neither varying nor non-varying, it is merely a string value. The attribute varying only applies to variable and is used to effect assignment to the variable.

This means that string expressions can be passed as arguments to either a varying or a non-varying string parameter.

In order to insure complete compatability with the standard programs which contain a returns attribute of the form returns (varying char(   )) must be changed to returns (char(*)). The Version II compiler will compile the old syntax and issue a warning diagnostic.

5. The alignment attribute of all scalar values must match the alignment attribute of the parameter to which they are passed. Version I only required the match for string scalars. This was an error in Version I and is being corrected to conform with the standard. An optimizing option can be used to cause aligned items to match unaligned items.

6. The Version I compiler has been issuing a warning diagnostic when the return attributes were not enclosed in the returns (   ) attribute of a procedure or entry statement. The Version II compiler will issue a fatal syntax error for this case.

7. The attributes: normal, abnormal, uses, sets, secondary, are no longer allowed. They were ignored by Version I.