Multics Technical Bulletin
Char*(*) Functions

MTB-581

To: Distribution

From: T. Oke & H. Hoover

Date: 04/20/82

Subject: Ansi77 Fortran Functions Returning Character*(*)

Comments should be sent to the author(s):

via Multics Mail:

Oke.Calgary, Westcott.Calgary, Hoover.Calgary, or
MPresser.Multics on System M.

via Mail:

M. E. Presser
Honeywell Information Systems
575 Technology Square
Cambridge, MA 02139
(617) 492-9320 or (HVN) 261-9320

T. Oke or H. Hoover or B. Westcott
The University of Calgary
2500 University Drive N.W.
Calgary, Alberta, Canada
T2N 1N4
(403) 284-6201

Multics project internal working documentation. Not to be reproduced or distributed outside the Multics project.

1 ABSTRACT

This MTB recommends a strategy for extension of the existing FORTRAN compiler to support the ANSI77 feature of 'character*(*)' functions. A 'character*(*)' function returns a character string of a length specified at invocation, by the caller. By contrast, the size of the return value of all other types of FORTRAN function is fixed at compile-time and independent of the caller.

2 PROBLEM

In FORTRAN, the name of a function is also the name of a variable local to that function and of the same type as the function. The value returned by the invocation of a function is defined to be the value, upon exit from the function, of the local variable of the same name as the function. Thus it is convenient to refer to the local variable whose name is the same as that of the function as the function value.

For functions of any type other than 'character*(*)', the amount of storage needed for the function value is known at compile-time. Thus the function value can be allocated in either static or automatic storage, in the compilation unit of the function, just like any other local variable. The amount of storage required for the function value of a 'character*(*)' function is not known until run-time, because it is specified by the caller. Thus, the storage needed for the function value cannot be allocated in either static or automatic storage within the function's compilation unit.

The current version of the FORTRAN compiler always allocates the function value in automatic storage. The only difficulty in adding 'character*(*)' support is: How do you allocate storage for the function value? The solution of this problem must not degrade the performance of regular functions and should not require recompilation of existing programs.

3 SOLUTION

The ANSI77 definition of FORTRAN is such that the language can be implemented with no need for run-time storage allocation. In particular, it is possible to implement 'character*(*)' functions so that storage for the function's return value is allocated at compile time. However, there are serious drawbacks to using this approach in the Multics implementation of FORTRAN. Therefore, we propose to allocate the function value of 'character*(*)' functions by a run-time extension of the function's stack frame.

We justify this proposal in three steps:

- First we explain how storage for the function value can be allocated at compile-time.
- Then we indicate the problems with this approach.
- Finally we show how run-time allocation by stack frame extension will overcome these problems.

According to the ANSI77 FORTRAN standard, the caller of a 'character*(*)' function must specify (with an unsigned integer constant or integer constant expression) how long a string will be returned. Thus the space for the function value of a 'character*(*)' function can be allocated (at compile time) in the caller, and passed to the function (at run-time) as a hidden parameter.

The caller of a character function cannot tell whether that function is a 'character*(*)' function or a fixed-length character function. Thus, if the above strategy were adopted, fixed-length functions must abide by the same calling sequence as 'character*(*)' functions. This is a problem, since we have existing programs that call fixed-length character functions.

If we implement 'character*(*)' function via the above strategy, we must either break some existing compiled programs or suffer a performance degradation in all fixed-length character functions: If we continue to allocate storage for the function value in fixed-length character functions, we won't break existing programs but we will have reduced efficiency in new programs, due to the extra overhead of copying the result into the caller's (unneccesary) temporary. If, on the other hand, we have fixed-length character functions use the caller's temporary to hold the function value (as in the 'character*(*)' case), some existing programs will break, since they will not have generated a temporary (so changing the function value may also illegally alter a parameter).

Another problem with the above strategy is in compatibility with other languages: Since they assume the same calling conventions for character functions as the current FORTRAN implementation, the result of calling a 'character*(*)' function from outside FORTRAN may be incorrect.

The above problem of choosing between efficiency and compatibility with existing compiled programs can be avoided by choosing a different method of implementing 'character*(*)' functions: have the function dynamically allocate the function value by extending its stack frame. This would make 'character*(*)' functions run slightly slower than with the above method. However, there would be no compatibility problems, and fixed-length character functions would run at the same speed as now.

It is slightly against the "flavor" of FORTRAN to do dynamic storage allocation, but this "flaw" is internal, hidden from the view of the users. Besides, there is already precedent in the current FORTRAN compiler for dynamic storage allocation: the restriction in ANSI77 FORTRAN that expressions involving string concatenation may not be used as parameters has been ignored. The length of the temporary to which the expression is assigned is not known at compile-time, so the temporary is generated at run-time by stack extension.

4 PERFORMANCE

The performance of the above method of 'character*(*)' functions should not suffer noticably with the implementation. The method of implementing the stack movement will be quite simple, with the size of the current stack frame, in the frame header, being incremented by the size of the string, determined from the descriptor. Additional fixed space may be required to hold a descriptor of the temporary, but this can be allocated on the stack at compile time, and filled in at run-time. On return from the function, the same code as is currently used will remain, with the length of the string for the EIS copy coming from the descriptor of the string.

5 OPTIMIZATIONS

It is not seen at this time that any form of optimization is possible, over and above the normal character function optimization. All the code which will be introduced by the extension of 'character*(*)' functions will be in the form of macro templates, which will be hand-optimized to run as quickly and be as short as possible.

6 DOCUMENTATION

The FORTRAN manual and reference quide, and appropriate info segments must be updated to indicate that 'character*(*)' functions are available.

7 HARDCORE SUPPORT

No hardcore support is necessary, all code is local to the FORTRAN system.

8 PROBE SUPPORT

Support of 'character*(*)' functions will require some form of descriptor to enable probe to find and print the intermediate value of the function during the execution of the function. This will require further information on the symbol table utilities and probe to fully determine what is necessary and how it may be implemented.

9 CROSS-LANGUAGE CALLS

As noted in the SOLUTION section, the above method has been chosen, not only to prevent the breaking of existing FORTRAN character functions, but also to retain compatability with cross-language calls.