To: J3
From: Larry Meadows
Subject: Unresolved issue 255

References: J3/97-153, J3/00-139

In the context of /interop discussions at the WG5 meeting in Oulu, unresolved issue 255 came up.

This paper proposes to resolve issue 255 via a revision of the command-line argument feature. The Revision is based on paper 00-139 that in turn was based upon 97-153. 97-153 discusses the rationale in detail.

At the time, one of the arguments against 97-153 was that it provided no simple way to retrieve arbitrarily long command-line arguments; independent changes have provided a clean resolution: Allocatable character lengths are now allowed and provide a simple means of addressing this issue.

The approach of the current f2k draft has several shortcomings:

1. It cannot be done anywhere except in the main program, thus forcing architectural decisions on the user. This is particularly problematic if the main program is not in Fortran or is otherwise not modifiable by the user who needs the command-line access. In addition this limitation is inconsistent with interoperability with C (that is, Fortran/C programs may well have a need to access the command line in places other than the main program, when coded in the most natural fashion).

2. It is not user-implementable, but can only be done by the compiler (largely a consequence of point 1). This means that users cannot do this with existing compilers.

3. It defines new language syntax and features, which invariably means new issues to get right. We probably haven't found all of them yet.

4. If extremely long command-line arguments are employed, the facility as currently defined is likely to be inefficient way to deal with them, as it forces all the command-line arguments to be returned in a single array. Thus if there is one argument of length 10,000 characters, and 1000 arguments of length 3 characters, you'll need a 10 megabyte array, about 1000 times larger than the actual data.

5. The approach of 97-153 is more flexible in that the user can declare the necessary variables in a way that fits the application. For example, they could be derived type components or the size could be allocated with extra room to accommodate requirements such as null termination (as mentioned in paper 00-121). With the current approach, these kinds of requirements require copying the data from the "pseudo dummy argument" to its preferred destination.

6. It is not at all similar to predominant existing Fortran practice.

It seems likely that the above shortcomings would cause a substantial fraction of users to continue to demand a procedure-based approach in addition to the approach of the standard.

Edits relative to 00-007:

{Delete program args from the bnf}

[235:14] Delete "[(<program-arg-list>)]

[235:15] Delete

[235:24-25] Delete

{Delete section 11.1.1}

[236:1-40] Delete

{Delete special case for char*(*)}

[68:25-26] Delete

{Delete special case for assumed shape}

[73:42-43] Delete "or...processor"

{These edits put the procedures in the ISO_FORTRAN_ENV module. It
 is also acceptable to make them intrinsic procedures in 13.16.
 The changes for such placement are trivial}.

[339:33] Insert after "."
  "It also provides procedures to access command arguments."

[340:22+] Insert new section

  13.17.3 Command arguments

  The module shall provide the following procedures for accessing
  command arguments.

  The meaning, interpretation, and means of providing command
  arguments are processor dependent.

  13.17.3.1 COMMAND_ARGUMENT_COUNT()

       Description. Returns the number of command arguments.

       Class. Pure function.

       Arguments. None.

       Result Characteristics. Scalar default integer.

       Result Value. The result value is equal to the number
       of command arguments available. If there are no
       command arguments available or if the processor does not
       support command arguments, then the result value is 0.

       If the processor has a concept of a command name, the
       command name does not count as one of the command arguments.

  13.17.3.2 GET_COMMAND_ARGUMENT(NUMBER [, VALUE, LENGTH, STATUS])

Description. Returns a command argument.

Class. Pure subroutine.

Arguments.

NUMBER shall be scalar and of type default integer. It is an INTENT(IN) argument.

It specifies the number of the command argument that the other arguments give information about. Useful values of NUMBER are those between 0 and the argument count returned by the COMMAND_ARGUMENT_COUNT intrinsic. Other values are allowed, but will result in error status return (see below).

Command argument 0 is defined to be the command name by which the program was invoked if the processor has such a concept. It is allowed to call the GET_COMMAND_ARGUMENT procedure for command argument number 0, even if the processor does not define command names or other command arguments.

The remaining command arguments are numbered consecutively from 1 to the argument count in an order determined by the processor.

VALUE(optional) shall be scalar and of type default character. It is an INTENT(OUT) argument. It is assigned the value of the command argument specified by NUMBER. If the command argument value cannot be determined, VALUE is assigned all blanks.

LENGTH(optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the significant length of the command argument specified by NUMBER. The significant length may including trailing blanks if the processor allows command arguments with significant trailing blanks. This length does not consider any possible truncation or padding in assigning the command argument value to the VALUE argument; in fact the VALUE argument need not even be present. If the command argument length cannot be determined, a length of 0 is assigned.

STATUS(optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the value 0 if the argument retrieval is successful. It is assigned a processor-dependent non-zero value if the argument retrieval fails.

One possible reason for failure is that NUMBER is negative or greater than COMMAND_ARGUMENT_COUNT().

Example.

```
Program echo
  use iso_fortran_env
  integer :: i
  character :: command*32, arg*128

  call get_command_argument(0, command)
  write (*,*) "Program name is: ", command

  do i = 1 , command_argument_count()
    call get_command_argument(i, arg)
    write (*,*) "Argument ", i, " is ", arg
  end do
end program echo
```

13.17.3.3 GET_COMMAND([COMMAND, LENGTH])

Description. Returns the entire command by which the program was invoked.

Class. Pure subroutine.

Arguments.

COMMAND(optional) shall be scalar and of type default character. It is an INTENT(OUT) argument. It is assigned the entire command by which the program was invoked. If the command cannot be determined, COMMAND is assigned all blanks.

LENGTH(optional) shall be scalar and of type default integer. It is an INTENT(OUT) argument. It is assigned the significant length of the command by which the program was invoked. The significant length may including trailing blanks if the processor allows commands with significant trailing blanks. This length does not consider any possible truncation or padding in assigning the command to the COMMAND argument; in fact the COMMAND argument need not even be present. If the command length cannot be determined, a length of 0 is assigned.