Subject: **Command Line Arguments ala Java**
Author: Jerry Wagener
Date: 11 Dec 1997

Java does command-line arguments in a nice, easy-to-use, almost-intuitive way:

```
class CommandLine             // note: “String” is the Java character-string type
{
    public static void main (String args[])
 {   . . .     //  args[0] is the value of the first command-line argument
     . . .     //                      (space delimited), args[1] the second, etc.
 }             //  args.length is the number of arguments
}              //  args[i].length() is the length of the ith argument
```

The Fortran analogy would be:

```
  program CommandLine (args); character(*) args(:)
    . . .      !  args(1) is the value of the first command-line argument
    . . .      !                        (space delimited), args(2) the second, etc.
    . . .      !   size(args) is the number of arguments
  end          !   len(args) is the length of the longest argument
```

In the example execution

```
      a.out  Las Vegas   143
```

size(args) would be 3, args(1) would have the value `"Las   "`, args(2) the value `"Vegas"`, and args(3) the value `"143   "`.

To accommodate command-line arguments in Fortran in this manner, the program statement is extended to optionally include a dummy argument list with one dummy argument; that argument must be declared as a rank-one assumed-shape assumed-length character array. The corresponding actual argument is supplied by the system upon execution of the program and comprises the system-tokenized (typically space-delimited) command-line argument character strings. The size of the actual argument is the number of such tokens in the command line, and the element length of the actual argument is (at least) that of the longest such token.

On the next page are the syntax and edits for incorporating this form of command-line arguments into Fortran; following that are the syntax and edits for an extended such facility with additional options. The latter "covers all bases", if that is deemed necessary, except that command-line arguments still must be routed through the main program. (Though, arguably, the main program is the "right" place to receive command-line arguments.)

Syntax and edits for the basic facility

R1102 *program-stmt* **is** PROGRAM *program-name* [(*command-args-array-name*)]

Constraint: a *command-args-array-name* shall be a rank-one assumed-shape assumed-length character array.

11.1.4 **Command-line arguments** {new section}

Command-line arguments are obtained in the program through the optional argument on the PROGRAM statement. This (dummy) argument is a character array that assumes its size and element length from the associated actual argument. The actual argument is provided by the processor upon invocation of program execution. The elements of the actual argument array are the system-tokenized (typically blank-delimited) character strings provided as command-line arguments at the time of program invocation. The first such token is the value of the first element of the actual argument array, the second is the second element of the actual argument array, and so on. The size (number of elements) of the actual argument is the number of tokens extracted from the command line. The length of each actual argument array element is no less than the number of characters of the longest such token; tokens are placed in the actual argument array elements in accordance with the rules of character assignment (that is, blank padded on the right if necessary).

Example:

program labAnalysis (**c_args**); character(*) c_args(:)
. . .
end

execution of program labAnalysis with the command: **a.out -if lab.data**

would result in:size(c_args) having the value 2
c-args(1) having the value `"-if      "`
c-args(2) having the value `"lab.data"`

Syntax and edits for a more extended facility

R1102 *program-stmt* is PROGRAM *program-name* [(*command-line-input*)]

R1102a *command-line-input* **is** *command-args-array-name* [*command-length-array-name*]
or *untokenized-command-line-name*

Constraint: a *command-args-array-name* shall be a rank-one assumed-shape assumed-length character array.
Constraint: a *command-length-array-name* shall be a rank-one assumed-shape integer array.
Constraint: an *untokenized-command-line-name* shall be an assumed-length scalar character variable.

11.1.4 **Command-line arguments** {new section}

Command-line arguments are obtained in the program through the optional argument list on the PROGRAM statement. The program may receive these command-line arguments either as an already-tokenized list of character strings or as a single untokenized character string, depending on the nature of the dummy argument list on the PROGRAM statement. When the (first) dummy argument is a character array, it assumes its size and element length from the associated actual argument. The actual argument is provided by the processor upon invocation of program execution. The elements of the actual argument array are the system-tokenized (typically blank-delimited) character strings provided as command-line arguments at the time of program invocation. The first such token is the value of the first element of the actual argument array, the second is the second element of the actual argument array, and so on. The size (number of elements) of the actual argument is the number of tokens extracted from the command line. The length of each actual argument array element is no less than the number of characters of the longest such token; tokens are placed in the actual argument array elements in accordance with the rules of character assignment (that is, blank padded on the right if necessary).

In those cases where one or more blanks at the end of a command-line argument string is significant, the optional integer array argument can be used to provide the system-supplied number of characters for each argument. The value of the first element of this array is the operable length of the first command-line argument character string, the second is the length of the second character string, and so on; the size of the integer array is the same as the size of the character array.

If a complete, untokenized command line is needed, the scalar character variable option is used.

Examples:

- - - - - - - - - -- -

```
program labAnalysis (c_args); character(*) c_args(:)
  . . .
end
```

execution of program labAnalysis with the command: **a.out -if lab.data**