Variable-Length String Input in Ada

Jeffrey R. Carter
Senior Engineer, Software
Martin Marietta Astronautics Group
P. O. Box 179
Denver, CO 80201

On a number of occasions I have found myself discussing how long a string needs to be declared for input, usually user input. Is twenty characters enough? Is eighty too many? Doesn't 256 use too much memory? This question comes up even when dealing with some form of variable string. The answer I always give is that the software should accept as many characters as the user enters, but have had to admit that declaring a string with POSITIVE'LAST characters is a little excessive. I felt that there must be some way to use Ada to create a string of just the right length, and have now figured out how to do that.

The basic concept is a recursive function which eventually returns a string of the length entered and skips the line terminator which marks the end of the string. This function result can be used to initialize a constant string or be passed as the actual parameter to a subprogram which has a string formal parameter of mode in. This function, which I call get_line, is:

```
with text_io;
function get_line (file : text_io.file_type := text_io.current_input)
return string is
      char : character;
begin -- get_line
      if text_io.end_of_line (file) then
            -- skip line terminator--ready to "get" next "line"
            text_io.skip_line (file);
            return ""; -- null string terminates recursion
      else
            text_io.get (file, char);
            return char & get_line (file => file);
      end if;
end get_line;
```

The default parameter value allows the function to be used without supplying an actual parameter. When this is done, the function reads from the default input file, which is the same file read by the text_io input subprograms which do not have file parameters.

The function can be used as described above; for example, a simple command-line interpreter might look like:

```
with text_io, get_line;
procedure command_line_interpreter is
      prompt : constant string := ">";

      logout_command    : constant string := "logout";
      copy_command      : constant string := "copy";
      delete_command    : constant string := "del";
      catalog_command   : constant string := "cat";

      procedure copy_file        (command_line : in string) is separate;
      procedure delete_file      (command_line : in string) is separate;
```

```
      procedure catalog          (command_line : in string) is separate;
      procedure execute_program (command_line : in string) is separate;
begin -- command_line_interpreter
      all_commands : loop
            text_io.put (prompt);
            get_command : declare
                  command : constant string := get_line;
            begin -- get_command
                  exit all_commands when
                        command'length >= logout_command'length and then
                        command (command'first ..
                                 command'first + logout_command'length - 1
                                ) = logout_command
                  ;
                  if command'length >= copy_command'length and then
                     command (command'first ..
                              command'first + copy_command'length - 1
                             ) = copy_command
                  then
                        copy_file (command_line => command);
                  elsif command'length >= delete_command'length and then
                     command (command'first ..
                              command'first + delete_command'length - 1
                             ) = delete_command
                  then
                        delete_file (command_line => command);
                  elsif command'length >= catalog_command'length and then
                     command (command'first ..
                              command'first + catalog_command'length - 1
                             ) = catalog_command
                  then
                        catalog (command_line => command);
                  else -- not recognized; assume it's an executable program
                        execute_program (command_line => command);
                  end if;
            end get_command;
      end loop all_commands;
end command_line_interpreter;
```

Apparently no one has described this approach to variable-length string input before. Much work has been done on variable-string abstract data types in Ada. This simple function might serve as an argument against the need for such a type, especially if it is being used only to facilitate input.

I would like to thank Geoff Mendal, technical editor of *Ada Letters*, for his valuable comments on earlier versions of this article.