A RT-11 FORTRAN PROMPTING LIBRARY

James Brakefield
Technology Incorporated
San Antonio, Texas

ABSTRACT

The paper presents a Fortran subroutine library that greatly facilitates programming the user interface. The prompting routines accept a default value, upper and lower limits, a prompt string, and the conversion radix. These routines prompt the user for a new value, validate and modify the value, and return a code reflecting the user response.

These routines are used for integer, real, yes/no, file name, and menu prompting. Subsidiary routines handle text to/from binary conversions. This allows application programs to be developed without the use of Fortran I/O or format statements (useful when memory is limited). SYSLIB is used for the RT11 interface and for 32-bit arithmetic.

The code returned by the prompting routines allows straight-forward control of the prompting dialogue. There are codes for "help", "skip", and "exit". The codes returned are as function values and can be ignored by using the subroutine call statement instead of function invocation. Typically, an application is first done with subroutine prompts. Once working satisfactorily, these are converted to function prompts and the appropriate help, skip, and exit code inserted. A glossary routine allows help requests to be directed to a disk-based glossary.

TALK

This text covers programming done by Technology Incorporated under contract to the Air Force at Brooks AFB, San Antonio, Texas, over a period of 1979 to the present. The computing facility consists of one or more PDP 11/34's and/or PDP 11/03's running RT11 versions 3.0 and up.

INTRODUCTION

Recent reports indicate that well-designed and comprehensive subroutine libraries are extremely economic. Programming productivity improvements of 300 to 800 percent are possible. Our experience confirms this. Our prompting library has very definitely simplified and sped the programming of the user interface. The resulting source code is readable and maintainable. The user interface is improved over traditional Fortran coding. It is both more consistant and bombproof.

(A change is occurring in the nature of libraries. They are now often called "modules" and encompass declarative information as well as procedural. The rationale is often described in terms of hiding implementation details of one level from the next higher level.)

Our experience is that some libraries have greater utility than others. The libraries of greatest utility tend to be device handlers of one sort or another. In this respect, the prompting library (which has the greatest use of all our libraries) could be considered as a device handler, where the device is the "user". As such it fits above the physical console device handler and below the application program. In our experience it is not unusual to have several levels of "device handlers".

HISTORY

Various versions of the prompting library have been in use for six years. The first version was done by Ms. Carman Galvan as part of our application programming. The customer wanted ease-of-use (i.e., a user-friendly system). This was back in the days of RT11 version 2 and 3. Basically, the user did not want to carry around a user's manual in order to use the software.

The first version used standard Fortran I/O (i.e., Format statements). Since we did not have any experience at writing prompting code, the subroutines were very ad-hoc. Subsequent versions have used SYSLIB instead of FORLIB to minimize memory requirements. The actual subroutines have changed names and picked up a more consistant calling format, and several "helper" routines have emerged.

Along the way several features have been tried. The first library did not use functions. Since then all versions have had the main prompting routines return a code to report the results of the prompt. The first set of codes was +1, -1, and 0. Minus

one was the error return. This was thought to fit well with Fortran IF statements. We then tried a 1 through 5 reply, which fits well with computed GOTO's. There is definitely a need for more than three reply codes. The ability to call a function as a subroutine (ignoring the reply codes) is handy.

The other feature tried was to use the radix as a flag. If negative, the prompting routine skipped the user dialogue. This was so one could go back into the middle of a prompting section, change a particular value, and skip to the end of the section. This feature had as many problems as benefits. With the expanded reply code, the negative radix was no longer needed. Some of the routines do not use a radix, so it was somewhat inefficient to require that it be included in the call.

Recently the file name prompting was simplified. Previously the dialogue asked if the file name was OK. If not, it prompted for the device, name, and extension on separate lines. Now the default file name is displayed. Any part may be changed separately. Thus a reply of DK1:.SAV will change the device and the extension but not the name.

Often the reply should be a symbol instead of a number. This is handled by passing a text string and indicies into this string to a prompting routine for this purpose. The current routine allows abbreviation via a second index string listing the minimum reply length for each symbol. This implements I/O for the enumerated type found in PASCAL but which is not normally available in Fortran.

The numeric routines are programmed to handle any radix from 2 to 36. Almost all of our usage is in radix 10. We have added simplified routines that work in radix 10 only.

Ever since RT-11 version 3, we have used chaining. For a single "program" to have three separate .SAV files is not unusual; one for prompting, one for data calculation, and one for display of results. We use chaining instead of overlaying. Our programs are subject to frequent changes, and chaining requires less detailed memory planning than overlaying. We have also avoided common areas for the published reasons. Currently, we have over 100 .SAV files. The software is being continuously expanded.

We handle this by breaking the program into application areas. A master program prompts for the application area and chains off to it. Each application area has dynamic chaining. A given .SAV program may have one or more menu items for that area. This program only chains-off when the user asks for one of the nonresident menu items. Each menu item is a single subroutine call. All of this is stylized and straight-forward to implement.

Each application area has its own chain package of default values. The master program saves and retrieves these from a disk file whose extension is nominally the user's initials. Thus, each user can have his default values preserved between runs.

CURRENT USAGE

The handout contains the viewgraphs that show the history of our prompting library and examples of its use. The first set lists the subroutines in each successive version of the prompting library. Each library has a slightly different name. Earlier versions will continue to work as long as the reply codes are not changed. Thus, there are three eras: the first library, which did not use reply codes; the +1, -1, and 0 era; and the current 1..5 era. Lately the number of routines in the prompting library has been growing faster than normal. In some cases there are several sets of equivalent routines to chose from.

The next set of viewgraphs shows the declaration section of several of the prompting routines. We follow the Pascal style with declarations for all subroutine parameters. Each variable is on a separate line, along with a comment describing that variable. In some cases the comment will extend for more than one line. The intent is that the declaration section give sufficient information for a programmer to be able to use that subroutine; thus, it is an important part of the documentation of that routine. In fact, we prefer to include as much documentation with the source code as is possible.

Within a library the routines are kept in alphabetical order. This is the most practical way to find something in the listing. It would be nice to group them according to area. If this is desired, it can be done in the "contains" section with a comment section outlining the subroutine groupings.

The BOOT.FOR listing is an example of simple usage of the prompting library. This is the program executed by the startup file. It makes sure that a time and date have been entered. It also records that time and date, so the most recent boot provides a set of default values (handy when rebooting several times a day). It also prompts for the user initials and chains off to the master program. The users quickly get familiar with programs they use every day. Thus, there is little need for help features or complicated logic. If they cannot get through this program, maybe they need assistance anyway; so the prompting routines are used without their function replies. A program like this goes together about as fast as you can type it in.

The last listing is an elaborate prompting example. It is built as a subroutine. All the many defaults are passed via the parameter list. Long parameter lists seem to work OK. They let you change either the caller or callee independently. This saves a lot of shuffling through listings.

Each use of a prompting routine is paragraphed, which makes the listing easier to read. The prompting is on the next line via a continuation. Any help information preceeds the prompt via the IGLOSS routine.

At the end of a prompting section, there is a large computed GOTO using the NYCHG function. NYCHG asks the user if he wants to repeat any of the prompts. The prompts are ordered A to Z. When the letter associated with the prompt is typed in, NYCHG will return the index associated with the letter and the computed GOTO will jump to the appropriate code. The user can then redo the prompt. Control then passes to the next prompt, where the user can use the "skip" reply to jump directly to the NYCHG. This time if the user answers NYCHG with just a

carriage return, the GOTO jumps to the next section.

People who do forms-type applications would probably use a "case" style of NYCHG. Due to the variety of consoles in use at our facility, we are pretty much locked into a scrolling dialogue. A forms style is generally done in the no-scroll mode. In the one application we did in a forms mode, the prompt was always at the bottom of the screen. The answering reply was posted to the "form".

All text is sent to the terminal with the PRINT subroutine. So that the cursor will stay at the end of the text, the zeros at the end of the strings are temporarily replaced by an octal 200. A routine called PUNT does this. It would be nice if SYSLIB had a variant of PRINT that did not append a carriage return--line feed to text strings.

All in all, we have attempted to add a feature (prompting) to a language (Fortran) in a fashion consistant with that language, and with the source code consistant with modern programming practice.

THE FUTURE

Several features could be added. The ability for IGLOSS to scan and search files would be useful. Rather than a whole series of short files, a single large file could contain the entire glossary. IGLOSS would search for a keyword and display text at that point. One could also scan other text files such as source code or data. This amounts to a windowing facility. Our constraint is the memory needed for file buffer and subroutine code.

One desirable feature would be the ability to input lists of things on a single line; another the ability to input sequences. These would call for a more elaborate input scanner. The demand has not yet been enough to justify the work involved.

Another desirable feature would be a symbol table. Numbers could be named, and the name later used to retrieve the value. This would be most helpful in naming data sets so that the scientist could save and retrieve his experimental data. Our problem with this is the size of our disk packs and memory limits. Larger disks and more memory are needed to make this viable.

To store sequences of computer processing and later command the computer to repeat the sequence would be nice. The prompting routine would need to be able to read a one line "program" and execute it. Here the symbol table, besides being a data base, would also be a repository of command sequences.

SUMMARY

The evolution of our prompting library over half a dozen versions has been towards a more refined appearance, both to the user and to the programmer. It is now at the embellishment stage.

The current package handles all user console dialogue. The coding of prompting sections is straightforward and is the least of our problems. Users have little trouble with applications software, even without formal training or user's guides. (Typically, users are on their own after being shown how to use an applications package once or twice.)

In the future well-integrated packages can be expected from the software vendors. Our preference is for packages that are provided subroutine library form. The lack of virtual memory on the 11's will make it difficult to do large, integrated packages, at least in our free-wheeling piece-at-a-time style.

REFERENCES

(1) Military Electronics, April 1984.

(2) PASCAL User Manual and Report, K. Jensen and N. Wirth, Springer-Verlag, 1974.

(3) On the Criteria to Be Used in Decomposing Systems into Modules, D.L. Parnas, Communications of the ACM, 15:12:1053-1058, 1972.

carriage return, the GOTO jumps to the next section.

People who do forms-type applications would probably use a "case" style of SYCHO. Due to the variety of consoles in use at our facility, we are pretty much locked into a scrolling dialogue. A forms style is generally done in the no-scroll mode. In the one application we did in a forms mode, the prompt was always at the bottom of the screen. The answering reply was posted to the "form".

All text is sent to the terminal with the PRINT subroutine. So that the cursor will stay at the end of the text, the zeros at the end of the strings are temporarily replaced by an octal 200. A routine called PUNT does this. It would be nice if SYSLIB had a variant of PRINT that did not append a carriage return--line feed to text strings.

All in all, we have attempted to add a feature (prompting) to a language (Fortran) in a fashion consistent with that language, and with the source code consistent with modern programming practice.

THE FUTURE

Several features could be added. The ability for IDLOSS to scan and search files would be useful. Rather than a whole series of short files, a single large file could contain the entire glossary. IDLOSS would search for a keyword and display text at that point. One could also scan other text files such as source code or data. This smounts to a windowing facility. Our constraint is the memory needed for file buffer and subroutine code.

One desirable feature would be the ability to input lists of things on a single line; another the ability to input sequences. These would call for a more elaborate input scanner. The demand has not yet been enough to justify the work involved.

Another desirable feature would be a symbol table. Numbers could be named, and the name later used to retrieve the value. This would be most helpful in naming data sets so that the scientist could save and retrieve his experimental data. Our problem with this is the size of our disk packs and memory limits. Larger disks and more memory are needed to make this viable.

To store sequences of computer processing and later command the computer to repeat the sequence would be nice. The prompting routine would need to be able to read a one line "program" and execute it. Here the symbol table, besides being a data base, would also be a repository of command sequences.

SUMMARY

The evolution of our prompting library over half a dozen versions has been towards a more refined appearance, both to the user and to the programmer. It is now at the embellishment stage.

The current package handles all user console dialogue. The coding of prompting sections is straightforward and is the least of our problems. Users have little trouble with applications software, even without formal training of user's guides. (Typically, users are on their own after being shown how to use an applications package once or twice.)

In the future well-integrated packages can be expected from the software vendors. Our preference is for packages that are provided subroutine library form. The lack of virtual memory on the 11's will make it difficult to do large, integrated packages, at least in our free-wheeling place-as-a-time style.

REFERENCES

(1) Military Electronics, April 1984.

(2) PASCAL User Manual and Report, K. Jensen and N. Wirth, Springer-Verlag, 1974.

(3) On the Criteria to Be Used in Decomposing Systems into Modules, D.L. Parnas, Communications of the ACM, 15:12:1053-1058, 1972.