1987-06

TERM IO: an Ada terminal interface package

Keough, Anthony James.

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

TERM IO:
AN ADA® TERMINAL INTERFACE PACKAGE

by

Anthony James Keough

June 1987

Thesis Advisor: Daniel L. Davis

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; Distribution is Unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b OFFICE SYMBOL (If applicable) Code 52	7a NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
TERM IO: AN ADA TERMINAL INTERFACE PACKAGE (u)

12 PERSONAL AUTHOR(S) Keough, Anthony James

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1987 June	73

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Terminal Interface; ANSI Standard Terminal; Screen Control Functions; Ada

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

One difficulty in the use of the Ada language in interactive programming is the inability to specify serial CRT terminal screen functions when writing the user interface. This thesis presents a solution in the form of an Ada package for terminal IO that provides the programmer with Ada language function calls that perform many of the serial CRT screen control functions automatically available in other languages. A specification of the package TERM IO is presented. An implementation of the package for the VT-100 terminal and an example of the use of TERM IO are presented.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Daniel L. Davis	22b TELEPHONE (Include Area Code) 408-646-3091 22c OFFICE SYMBOL Code 52Dv

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

Approved for public release; distribution is unlimited.

TERM_IO:
An Ada Terminal Interface Package

by

Anthony James Keough
Lieutenant, United States Navy
B.S.M.E., University of Wisconsin, 1981

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL

June 1987

ABSTRACT

One difficulty in the use of the Ada language in interactive programming is the inability to specify serial CRT terminal screen functions when writing the user interface. This thesis presents a solution in the form of an Ada package for terminal IO that provides the programmer with Ada language function calls that perform many of the serial CRT screen control functions automatically available in other languages. A specification of the package TERM_IO is presented. An implementation of the package for the VT-100 terminal and an example of the use of TERM_IO are presented.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

The ability to understand and use a computer software product depends heavily on the quality of the user interface. The user interface is the vehicle of conversation between the programmer and the user. The programmer wants to provide a clear. easy to use interface so that his work is favorably evaluated and used. The user wants an interface that he can understand and use easily. Programmers employ a variety of techniques in the design of the user interface. The techniques used depend greatly on the type of hardware on which the the program runs and the capability of the language used by the programmer. Many languages provide a wide variety of functions that control the terminal interaction process while other languages are sparsely equipped. Often the amount of effort that is required of the programmer determines the quality of the user interface that he produces.

Thesis students at the Naval Postgraduate School (NPS), who are sponsored by the Naval Weapons Center, China Lake (NWC) and who are programming in Ada have experienced difficulties in controlling the output to a serial CRT terminal screen in the Ada language. These problems are manifested in the areas of cumbersome user interface modules and non-portable interface modules. These

problem areas at first seem to be separate but both are symptoms of the lack of terminal interface capability in the Ada language.

1. User Interface Modules

The user interface of the Ada programs being developed today at the Naval Postgraduate School are cumbersome, hard to use, and do not fully utilize the capabilities of the user's terminal. Programs such as AdaMeasure [Ref. 1] were written using the standard Ada packages text_io and serial_io which were designed to be used on teletype terminals. The program is usually run from a VT-100 terminal with much more capability. The user is not able to use the capabilities of his equipment to operate the program as easily.

2. Lack of Portability

Programmers who try to use the capability of the terminal to improve their program's interface often sacrifice portability. The AdaMeasure program discussed above uses a single VT-100 control sequence to clear the terminal screen, making the program non-portable. A programmer wishing to adapt the program to a different terminal type is required to research the method of clearing the screen and adapt the control sequence in the program.

This thesis examines some of the problems of writing a good interface in Ada and proposes a solution by providing a package of interface procedures. The programmer can use this package to better exploit the capabilities of the serial CRT terminal and enhance the quality of the user interface that he designs.

B. ORGANIZATION

Chapter Two describes the styles of user interfaces that are used today. The effect of the hardware capabilities on the choice of the user interface style is discussed.

Chapter Three discusses the design of the Ada language and the reasons for the lack of user interface procedures. The experiences of students programming in Ada and the need for a more sophisticated interface package is presented.

Chapter Four discusses the capabilities available in the common terminal types and how other popular languages have used the capabilities of the serial CRT terminal. An examination of these features helps to determine what procedures might be useful in an Ada interface package.

Chapter Five discusses the issues considered in the design of an improved interface package. The design of the TERM_IO interface package is presented and its capabilities discussed.

Chapter Six discusses the implementation of the user interface package body. An application of the package of interface procedures is demonstrated using a sample menu producing procedure.

Chapter Seven presents a summary of the work discussed and recommendations for the use of the TERM_IO package and for future work in this area.

Appendix A lists the complete specification of the package TERM_IO. Appendix B contains a programmers guide to the use of the TERM_IO package.

Appendix C lists the package body of the TERM_IO package for the VT - 100 terminal. Appendix D presents a summary of serial CRT control codes for several terminal types. Appendix E gives examples of the use of the TERM_IO package to improve the interface of an existing Ada program, the AdaMeasure program written by thesis students at the Naval Postgraduate School.

II. THE USER INTERFACE

A. INTRODUCTION

The purpose of the user interface is to bridge the conceptual gap between the user's understanding of the computing process and the program in execution. The user is not concerned with the hardware implementation details, only with the functional process of the program that is running.

The capability and functions of the user interface vary greatly from program to program. Some variation in capability is due to the decisions of the software designers. Many differences, however, are due to the constraints imposed by the capabilities of the supporting terminal.

B. TYPES OF TERMINALS

Terminal hardware configurations are different for every system. The capabilities of the terminal depend on the level of sophistication of its hardware. These capabilities vary from the basics of a teletype printer to the latest design of graphics terminal. The common terminal types in use today are the teletype, serial CRT, character mapped CRT, and the bit mapped CRT.

1. Teletype Terminal

The teletype terminal is the first widely used terminal. Many are still in use today. In this terminal the computer output is sent one character at a time in

serial fashion from the computer to be printed on the paper of a printer which sits behind a keyboard. The input is taken from the keyboard and is usually echoed on the paper. The user may address the memory by line only. The teletype has the least capability of any terminal type.

2. Serial CRT

The serial CRT terminal is an improved version of the teletype terminal. The paper and printer have been replaced in this system by a CRT screen that displays the output printed by the computer. This terminal has several advantages over the teletype including speed and paper saving. The only capability that the serial CRT terminal adds is that it is cursor addressable. The computer can position the cursor to any position on the CRT screen to print the next character. This feature can be used to format the screen output and produce simple shapes and graphs. The serial CRT is probably the most common terminal in use today. The input device has not changed from the teletype, it is still the keyboard input.

The terminal controls what is displayed on the CRT screen by storing the screen contents in an array corresponding to the screen locations on the screen. A common size for this array is 24 rows by 80 columns. Each location holds information that indicates the character to be displayed at its screen position. Also stored is information such as highlighting and character and background shade. Input to the terminal is received just as in the teletype terminal. The stream of serial output from the computer is written to the screen buffer. The

video processor reads the screen buffer and uses a character generator to produce the characters on the screen. The video processor typically does this 30 times each second. In the serial CRT the screen buffer is not accessable to the program.

3. Character Memory Mapped CRT

In the character memory mapped CRT terminal, the method of storing the information to be displayed on the screen has changed to provide access to the character stored at each screen location. The output is the same as for the serial CRT. The contents of the memory storage locations in the screen buffer can be read and used by the program. The video processor works the same as for the serial CRT.

4. Bit Memory Mapped CRT

In the bit memory mapped CRT terminal the screen buffer has been expanded to use a memory location for each pixel location on the screen. A typical screen size 1024 by 768 pixels. This terminal is pixel addressable. This allows the drawing of more complex graphics than is possible with character based systems. In this terminal, the video processor reads the memory location corresponding to each pixel to determine the shade. The character generator is no longer used in the screen refresh cycle. Pointing devices such as a mouse or track ball are now included as input devices as well.

C. TYPES OF INTERFACES

General purpose interfaces today are divided into two main categories, hierarchical interfaces and event driven interfaces.

1. Hierarchical Interface

The hierarchical interface is commonly used with the teletype, serial CRT and character mapped CRT terminals. Systems designed using the hierarchical interface are usually menu driven. Using this type of system, the user makes a selection from each of a series of menus until reaching a functional level where a process is performed. After the process is completed, control of the program is be returned to the top or 'main' menu or to the last menu reached before the process execution began.

The hierarchy of the menu selections reflects the structure of the program. At each menu level. the user restricts his view of the program to the options left under that selection. If, after a process is performed, the user desires to run another process, the user must retrace the menu tree to reach the menu selection for the new process. A user can experience problems in working through several menu levels and remembering what options are available from any of the menu levels.

2. Event Driven Interface

Event driven systems usually use the bit mapped CRT terminal with a pointing device. The user can access most of the program functions directly from the terminal screen.

Functions are selected with the pointing device from menus that are always visible to the user. From almost every function, the user returns to the main screen. The prompt for the available functions is visible at all times so the user has less difficulty remembering what functions are available and how to access them.

This thesis concentrates on the user interface requirements of the hierarchical interfaces using serial CRT terminals. The systems currently available at NPS and NWC for Ada programming are all serial CRT devices. Almost all of them are VT-100's or VT-100 emulators.

III. THE ADA USER INTERFACE

The Ada language is designed for use in embedded systems software. Interfaces for embedded systems tend to be specialized and hardware specific. Little effort has been made to develop an interface for output devices other than the serial (embedded) device and the teletype terminal. As a result, programmers who want to write in Ada cannot use terminal interaction and screen formating available on the serial CRT terminal without writing their own interface procedures.

Most of the current Ada literature concerns the design of software engineering environments for the language. Many of these systems are being designed and written but none has specifically addressed the problem of using terminals with more capability than the original teletype. It is assumed that each application will write its own interface.

The Preliminary System Specification of the Software Technology for Adaptable Reliable Systems - Software Engineering Environmets (STARS-SEE) program specifies that a standard user interface at the end product level will be used. The Interface Control Working Group (ICWG) is responsible for the oversight and control of system and software interfaces[Ref. 2]. The preliminary specification does not address the variations of terminal types but only says that the interface is to be consistent.

The shortcomings of the TEXT_IO package were the subject of a paper by J.P. Rosen presented at the 1984 IEEE Ada Applications and Environments Conference [Ref. 3]. This paper describes typical programming problems with TEXT_IO. many of which are the same problems experienced by programmers at NPS. Rosen offers several programming techniques to better use the facilities in TEXT_IO. Rosen takes the position that a special IO package is not necessary to write a good terminal interface in Ada. He does not address the problems of using screen graphics or different terminal configurations.

While it is likely that any large commercial application written in Ada would include its own terminal IO package, the use of Ada in the academic and research environments would be enhanced by the availablility of a simple, easy to use, portable interface package that supported the serial CRT terminal.

Students at the Naval Postgraduate School are currently using Ada for general purpose projects other than embedded systems. Many of these projects use terminal interface and interactive procedures. These students and other programmers who want to use Ada for programs with terminal interface procedures face several difficult problems. Methods of performing many common screen functions such as clearing the screen or positioning the cursor are not entirely standardized. Students who want to improve the usability and appearance of their work have been forced to include terminal specific character codes and functions in their programs. The result is that each student is spending

time researching the correct character codes and is producing programs that are not portable.

The solution for these programmers is the availability of a package of terminal interface procedures that a programmer can use for terminal interaction. The objective of the terminal interface package is to provide the Ada programmer with the enhanced terminal interface functions required to exploit the capabilities of the serial CRT terminal. The programmer can use these functions without having to rewrite them for each program. The result will be more efficient, standardized programs.

The system should provide an interface between the program and the terminal device so that the programmer can specify the terminal functions by using standard procedure calls in his programs in the same way that text_io and serial_io procedures are called. The call will remain the same, but the implementation will depend on the terminal type.

IV. SURVEY OF EXISTING SYSTEMS

The Ada language, other language interfaces, and terminal manuals were studied to determine how user communications in the Ada language could be improved. Several existing commercial software language packages have features that fully utilize the capabilities of the serial CRT. These languages are implemented for the particular language on a particular mainframe or microcomputer. They indicate what features have been successfully implemented and what capabilities an Ada programmer is likely to utilize. Programmer's guides for the different terminal types indicate what functions can be implemented for each terminal type and the method to be used. By comparing the functions available on the terminals, the functions provided in other systems, and the experience of Ada programmers, a common set of useful functions that a programmer is likely to use can be chosen.

A. THE ADA LANGUAGE

Ada is designed to allow users to tailor the language to their needs by writing packages that can be used in many programs. The designers of the language purposely left out many of the terminal interface functions to maintain the overall generality of the language[Ref. 4,p.252]. Implementation of these functions has been left up to the user.

The standard text_io Ada package allows the programmer some, but not much, control of the screen output. Some of the procedures specified include: line and col that return the line or the column of the present cursor position, set_line and set_col that change the cursor position, and a new page function that advances the page[Ref. 5]. These procedures are designed to control the output of the teletype terminal and do not provide the level of access to the operating system functions that is required to control the output to a serial CRT terminal. In order to produce the terminal screen output, the programmer must specify the character to be printed at each screen location. In such a case, the CRT terminal screen is no more useful than the teletype printer paper it replaced.

B. OTHER LANGUAGE SYSTEMS

Languages that are designed to run on a particular system can provide a fairly extensive set of terminal control functions because the hardware configuration is known and portability is not an issue. This is particularly true of systems that run on microcomputers.

One such microcomputer language is Turbo Pascal. Turbo Pascal provides a set of Pascal procedures that the programmer can use to control the screen output. These procedures allow the programmer to clear a line of text, clear the screen, position the cursor, and adjust screen brightness. Turbo Pascal uses the ANSI escape codes and operating system calls of the host machine to accomplish these functions. [Ref. 6]

C. TERMINAL DOCUMENTATION

Documentation such as the VT-100 User Guide is available for most terminal types. This documentation provides the user with information about the terminal's capabilities and functions. The VT-100 User Guide summarizes the protocol of over fifty terminal features [Ref. 7,p.43]. Norton's programmer's guide to the IBM PC is an excellent source of information for users of IBM and MS-DOS microcomputers [Ref. 8]. The challenge to the programmer is to incorporate the terminal functions into the language that he is writing in, in this case Ada. The difficulty to the programmer is the need to devote significant time and effort to the ancillary problem of the user interface.

An examination of the code written by programmers for their interfaces illustrated the problems experienced by those programmers and what they would likely do if provided with a set of user interface procedures. Neider and Fairbanks have written their interface on the level of the teletype terminal without making use of the capabilities of the terminal [Ref. 1]. This is a 'lowest common denominator' approach. This method is hard on the user who may be used to working on other more advanced or friendly systems, such as Unix. Other programmers have spent hours trying to reproduce a function that they know is possible to do in another language. An excellent example of this is a keyboard polling function, that is a function in Turbo Pascal [Ref. 6,p.143], and in the 'C' language is the 'getchar' function. A keyboard polling function is not provided in Ada.

A common method of screen control seen in serial CRT terminals is the use of special character codes such as the ANSI code [Ref. 7]. The put function in text_io package is capable of writing these control codes to the terminal screen. By using the standard functions in the text_io package and the ANSI codes for the host machine, a useful set of terminal user interface procedures can be implemented.

V. DESIGN ISSUES

A. OPTIONS

By looking at the existing software and discussing the needs with programmers, the need for a basic set of terminal interface procedures has been established. The next step is to design an implementation of these features that is useful to the programmer and allows the programmer to write terminal interface functions easily.

Three alternatives were considered for the implementation of the terminal interface package. The low end alternative is to distribute a table of terminal functions and control codes to programmers to include in their programs. This method has several disadvantages. It requires strictly enforced programming standards to prevent giving the programmers too much discretion over how to use the control codes in their programs, promoting the non-portablity of code. It would defeat the organization's goal of portability and maintainability and send the 'wrong signals' to the programmers.

The middle alternative is to develop an Ada package that implements a set of functions that are common to many terminal types and restrict the programmers to using the package to perform terminal interaction. This will allow the code that is produced to be portable to any terminal type that is supported by a version of the terminal interface package. This method achieves the most

portability and standardization but it does not fully utilize the capabilities of each terminal type.

The high end alternative is to write a separate package for each terminal type that provides a full implementation of the capabilities of that terminal. This allows the best utilization of the terminal capabilities, but it has many of the same disadvantages as the low end option. Code will not be as portable. There will be no standardization, and more problems in maintenance of a larger set of programs.

Since many of the processes to be implemented are accomplished through the use of character sequences that vary from terminal type to terminal type, an Ada package can be used to declare the constants for a terminal type. The same set of procedures can be used, but with a different set of constants for each terminal type supported. This approach allows a package to be designed for any function that is performed by writing a special character sequence to the screen.

The choice of terminal types for implementation of the TERM_IO package was based largely on the types of terminals at the Naval Postgraduate School and the Naval Weapons Center, China Lake. The most common terminal is the VT - 100 terminal. The VT-100, an ANSI standard terminal, is the terminal used for current Ada projects. Most serial CRT terminals in use today can emulate the ANSI standard terminal to some degree.

The TERM_IO Package provides much more screen support than keyboard support. Keyboard functions are much more hardware oriented than the screen

functions and are less likely to be portable. The interface between the program and the screen functions is well defined by the use of character strings as terminal commands while the interface between the program and the keyboard function is not as well defined. The terminal screen usually accepts information in a character format but the keyboard can send information in character, integer, or other data format. The program must allow for different data types if keyboard polling is implemented extensively.

The terminal assumed for the TERM_IO package provides a terminal CRT screen of 80 columns by 24 rows, support for the basic ASCII character set, and a graphics character set extension that allows the graphics characters to be declared as character constants in Ada. Terminal control codes are available to perform the functions of cursor movement, screen and line clearing, reverse video and bold face printing, and graphics character printing. Other terminal features such as user controlled cursor keys, numeric keypads, and special function keys are not used in the TERM_IO package.

B. THE DESIGN

The package TERM_IO includes screen handling procedures that are not available in any of the standard Ada packages. These procedures allow the programmer to perform screen control functions in the Ada language. The package specification has been designed to be portable among many terminal types, provided that the package body has been modified for the terminal type in

use. The complete specification of the package TERM_IO is listed in Appendix A. Portions of the package declaration are reproduced and described below. Appendix B provides a programmer's guide with more discussion and examples of the use of the TERM_IO package.

```
with TEXT_IO;
use TEXT_IO;
package TERM_IO is

type SWITCHTYPE is (ON,OFF);

SWITCH : SWITCHTYPE;
```

The package TERM_IO uses the input and output procedures contained in the standard Ada package TEXT_IO. TEXT_IO should always be available in the Ada programming environment. The type SWITCHTYPE is an enumerated type of (ON,OFF). The variable SWITCH is a status variable used by of the text printed on the screen.

1. Cursor Control

The cursor control procedures can be used to position the cursor anywhere on the terminal screen. These are:

```
procedure MOVE_CURSOR_HOME;
procedure MOVE_CURSOR_UP(NUM : in integer);
procedure MOVE_CURSOR_DN(NUM : in integer);
procedure MOVE_CURSOR_RT(NUM : in integer);
procedure MOVE_CURSOR_LT(NUM : in integer);
procedure SET_CURSOR_POS(COLM,ROW : in integer);
procedure GET_CURSOR_POS(COLM,ROW : out positive_count);
```

In a program, a procedure call of MOVE_CURSOR_LT(3); moves the cursor three spaces to the left on the terminal screen.

2. Screen and Line Clearing

Procedures to clear the screen and to clear individual lines have been included. These procedures are called without parameters. These are:

```
procedure CLEAR_SCREEN;
procedure CLEAR_LINE;
procedure CLEAR_CURSOR_TO_EOL;
```

Procedure CLEAR_LINE clears the line the cursor is on. Procedure CLEAR_CURSOR_TO_EOL clears the portion of the line to the right of the cursor. Procedure CLEAR_SCREEN clears the screen but does not change the cursor position.

3. Keyboard Polling

Keyboard polling functions are used to get a single character from the keyboard, usually as a response from the user. These functions are called without parameters. They are:

```
function GET_KEY return character;
function KEYPRESSED return boolean;
```

The function GET_KEY can be used to get a response such as a menu selection. The function KEYPRESSED can be used to have the user signal readiness to continue.

4. Reverse Video and Boldface

Reverse video and boldface printing can be obtained using these procedures. All are called without parameters. They are:

```
procedure SET_REVERSE(SWITCH : in SWITCHTYPE);
procedure GET_REVERSE_STATUS(SWITCH : in SWITCHTYPE);
procedure SET_BOLD(SWITCH : in SWITCHTYPE);
procedure GET_BOLD_STATUS(SWITCH : in SWITCHTYPE);
```

The variable type SWITCHTYPE is an enumerated type of (ON,OFF). A procedure call of SET_REVERSE(ON); causes all printable characters printed on the terminal screen to appear in reverse video until a procedure call of SET_REVERSE(OFF); returns the output to to the normal mode. Bold face print works the same way.

5. Graphics

Simple graphics characters can be used to make a display better looking and more understandable. Primitive characters have been provided as well as several procedures. These procedures are:

```
procedure PUT_TOP_LT_CORNER;
procedure PUT_TOP_RT_CORNER;
procedure PUT_BOT_LT_CORNER;
procedure PUT_BOT_RT_CORNER;
procedure PUT_HORZ_BAR;
procedure PUT_VERT_BAR;
procedure PUT_CROSS;
procedure PUT_TOP_TEE;
procedure PUT_BOT_TEE;
procedure PUT_LT_TEE;
procedure PUT_RT_TEE;
procedure DRAW_BOX(COLM,ROW : in integer);
procedure DRAW_HORZ_LINE(LENGTH : in integer);
procedure DRAW_VERT_LINE(LENGTH : in integer);
```

Procedure DRAW_BOX draws a box centered on the terminal screen with the upper left corner at position (COLM,ROW). The graphics character procedures PUT_TOP_LT_CORNER through PUT_RT_TEE print a single graphics character to the screen and leave the cursor on the position of the character just printed. DRAW_HORZ_LINE and DRAW_VERT_LINE draw a line from the current cursor postition of of the length specified. A horizontal line is drawn from the cursor position to the right. A vertical line is drawn from the cursor position up the screen.

VI. IMPLEMENTATION

The package body of TERM_IO implements the portable TERM_IO package specification. The goal in writing the package body was to produce a package body for one terminal type that could be easily modified for other terminals. Terminal specific items were declared as constants or separate procedures that could be easily modified. The complete package body for TERM_IO implemented for the VT - 100 terminal is contained in Appendix C.

A. THE TERM_IO PACKAGE

The major part of the TERM_IO package body is portable. The few parts that are not deal with specific hardware of the terminal. These are the control codes and the method of keyboard input modes available in the terminal.

1. Control Codes

Control Codes are character sequences that are interpreted by the terminal as commands. These codes are used to perform terminal functions such as clearing the screen and moving the cursor. Control codes vary from terminal to terminal and are not compatible. There is an ANSI standard for terminal control codes, however, which many terminals are capable of emulating. The control codes for two common terminals, the ANSI standard and the VT - 52 terminal, are contained in Appendix D.

Portability is achieved by restricting the programmer from direct access to these control codes. Instead, the codes are used by the procedures available to the programmer through the package declaration. The control codes for the terminal are declared at the head of the package body. These codes are declared as string constants. The string constant can be referenced by name to avoid problems with embedded constants. The constants were used in these procedures instead of the actual code strings to enhance the portability of the package body.

2. Keyboard Input

When writing interactive programs, it is often required that the user select a menu choice or 'press any key to continue' reading an information screen. These user interactions are usually done with a 'keypress' routine that detects when a key has been pressed on the keyboard. In Turbo Pascal, this function is called 'KeyPressed'[Ref. 6, p.143]. This feature allows a user to move quickly through a hierarchy of menus with as few keystrokes as possible. Unfortunately, there is no keypress routine in the Ada language.

The keyboard input functions in the standard TEXT_IO package are the 'get' functions. These functions are used to get character, string, and number input from the user. These inputs are required by the Ada language to end in a terminator character, an end of line, or end of file. The Ada get procedures interpret the carriage return as the end of line or file. This means that with the standard get procedures the user must hit a carriage return after each menu selection. Since most other menu systems do not require these carriage returns,

the carriage return action becomes annoying to the user. This is important enough to attempt an implementation of a keypress function for the TERM_IO package.

The method of getting input from the keyboard is different for each implementation. Various methods of system dependent keyboard polling can be developed. The poller could be a system call or an interupt. It could be a pragma, a compiler interface command, to a language such as 'C' that already has a polling function. In the package TERM_IO, the function KEYPOLLER is not fully implemented. Instead, it has been stubbed with a a simple Ada 'get' procedure. The package can be used in this form or the 'get' procedure can be replaced with a hardware specific keyboard polling routine. Each system can install their own keyboard poller routine if it is desired. If the KEYPOLLER function is used with the get procedure, carriage returns are required.

The method of keyboard polling has been hidden in the private procedure KEYPOLLER. The programmer cannot use keypoller directly. It is called from the functions GET_KEY and KEYPRESSED.

The procedure GET_CURSOR_POS is designed to read the cursor position maintained by the terminal and return the row and column to the program. This function was not implemented due to problems involved in translating the row and column information that is provided by the terminal into an Ada variable. This procedure can only be implemented by a hardware specific call to the terminal.

B. TERMINALS

While the package declaration of the package TERM_IO is portable, the implementation of the package body is specific for each terminal type supported. The specifics of the implementation are hidden from the programmer in the package body. The programmer is aware of only the declaration, or calling statement for the procedures conatained in the package body. An implementation has been written for the Digital Equiptment Corporation VT -100 terminal. This type has been chosen because it is the most common and almost all comercial terminals are capable of emulating it. Modifying the package to use on another terminal type can be done with changes to the package body. The complete package body for TERM_IO implemented for the VT -100 terminal is contained in Appendix C.

1. The ANSI Terminal Interface

The VT - 100 terminal uses the control sequences established by the American National Standards Institute for controlling serial terminal screen output. These sequences all begin with the 'escape' character, 033H, and thus are known as 'escape codes.' The escape character can be written in Ada as 'ASCII.ESC.' A summary of the escape codes for the ANSI terminal are contained in appendix D. An MS - DOS microcomputer can use the ANSI codes with the device driver ANSI.SYS configuration.

To use the escape code sequences in Ada the codes must be declared as string constants. These string constants can be manipulated as any other strings

are in Ada, including writing them out with a put procedure from the TEXT_IO package. When the terminal receives the output string, it interprets it as a screen command which is executed. The string is not printed on the screen. An example of such a string declaration is

```
UPCRSR   :   constant string := (ASCII.ESC,'[','1','A');
```

This string causes the cursor to be moved up one line. The complete list of string declarations for the package body of TERM_IO is contained in Appendix C.

2. The VT-52 Interface

Another common terminal type is the Digital Equipment Corporation VT - 52 terminal. The VT-52 terminal also uses control sequences to control the screen. These sequences are in a different format and are not compatable with the ANSI sequences. These sequences can be written in Ada as string constants just as the ANSI sequences can be. The package body for the VT-52 interface would be the same as the package body for the ANSI interface except for the different declaration of the string constants. An example of a string declaration for the VT - 52 terminal is:

```
UPCRSR   :   constant string := (ASCII.ESC,'A');
```

This string would cause the cursor to be moved up one line.

A complete implementation of the package body could not be made for the VT - 52 because the VT - 52 does not provide all of the control functions

provided by the ANSI standard terminal. A partial implementation with only the common functions of the terminals is possible for use on the VT - 52.

3. Other Terminal Types

The TERM_IO can be adapted to other serial CRT terminal types by substituting the correct control codes for the new terminal type. If the terminal type uses a method other than control sequences to control screen functions, then revision of the individual procedures of the TERM_IO package body will be required. This revision is likely to result in a very hardware specific package body. It should still result in a package body that completely implements the TERM_IO package specification.

VII. CONCLUSIONS

As use of the Ada language becomes more common and more programmers are trained in its use there will be more efforts to utilize the portability and code reusability features of Ada. The TERM_IO package is one example of the use of these capabilities. The use of the Ada language to allow programmers to write programs that can be reused in a variety of different situations is a major strength of Ada and should be exploited whenever possible.

A. REVIEW

The need for an Ada package to provide programmers with procedures to control the serial CRT terminal was indicated by the problems experienced with the standard Ada package TEXT_IO. The package TERM_IO was developed to meet this need. The goals sought in the design of the TERM_IO package were reusability, portability, and ease of application. These goals were met through the use of the constructs of the Ada language that allowed the terminal specific items of the program to be hidden from programmer's using the package.

This thesis has proposed a package specification for an Ada terminal interface package that is reusable and portable. An implementation of the features of the package and the method for implementing the package on other terminal types

has been presented. This package can be used by other programmers to design better user interfaces faster and easier.

The package TERM_IO was written to be used in the same manner as the standard package TEXT_IO and can be thought of in the same way by the programmer. The TERM_IO package provided the programmer with cursor control capability to format screen output, with parameter control procedures to change the characteristics of the output text, with input procedures to poll the user for input, and with simple character graphics capabilities to improve the appearance of the screen output.

B. RECOMMENDATIONS

The package TERM_IO should be provided to programmers learning Ada. It can serve as a useful tool for program development and as an example of a reusable package. Many of the problems experienced by the programmer who is new to Ada but has experience in other languages result from the inadequacies of the TEXT_IO package. A programmer who feels that he is always 'reinventing the wheel' to do screen output might well find a use for TERM_IO.

The TERM_IO package was designed for use with the serial CRT terminal. As Ada applications become available for bit mapped graphics terminals, there will be a need for a terminal interface package that provides a complete set of graphics functions for the Ada language. This area should be considered for further thesis research.

This theses was undertaken with the sponsorship of the NWC Missile Software Group as part of a continuing program. The TERM_IO package has been applied to improve the interface capabilities of other past and present thesis efforts and it will be available for future efforts.

Work on this and other theses in Ada for NWC has been made difficult by the lack of an Ada compiler for the Computer Science Department computer at the Naval Postgraduate School. Currently the school's only capability in Ada is the Janus/Ada partial implementation in use on microcomputers at the school. Working in the full Ada language required the use of the Telnet or Arpanet system to work on the China Lake computer system. under a Missile Software Group account. With the increased use of the Ada language for both thesis research and class projects at the Naval Postgraduate School, the acquisition of an Ada compiler should be considered.

The user interface is an important part of the overall programming effort. Efforts such as TERM_IO make the writing of the user interface faster and easier for the programmer. The hope is that the programmer will make use of the TERM_IO package to produce a well designed and easy to use interface.

APPENDIX A

TERM_IO PACKAGE DECLARATION

```
--************************************************************
--
-- TITLE:        ADA TERMINAL INTERFACE
--
-- MODULE NAME:   TERM_IO DECLARATIONS
--
-- Date created:  04 MAR 87
-- Last modified: 15 MAY 87
--
-- AUTHOR:       LT Anthony J. Keough
--
-- DESCRIPTION:  This package provides procedures to
--      improve the terminal interface. It should be
--      used with the package TEXT_IO to provide a
--      full set of user interaction procedures.
--
--
--************************************************************

with TEXT_IO;
use TEXT_IO;

package TERM_IO is

   -- To use TERM_IO the standard output must be set to
   --    the terminal screen.


  -- Variable types:

  type SWITCHTYPE is (ON,OFF);


  -- Variables:

  SWITCH : SWITCHTYPE;
```

```
-- Cursor control procedures:

  procedure MOVE_CURSOR_HOME;
     -- Positions the cursor to the top left position.

  procedure MOVE_CURSOR_UP(NUM : in integer);

  procedure MOVE_CURSOR_DN(NUM : in integer);

  procedure MOVE_CURSOR_RT(NUM :in integer);

  procedure MOVE_CURSOR_LT(NUM : in integer);

  procedure SET_CURSOR_POS(COLM,ROW : in integer);
     -- Positions cursor to the screen position (COLM, ROW)
     --    where (0,0) is the upper left corner.

  procedure GET_CURSOR_POS(COLM,ROW : out integer);
     -- Returns the screen position of the cursor where (0,0)
     --    is the upper left corner.
     -- Not implemented, stubbed to return (0,0).


--Screen and line clearing procedures:

  procedure CLEAR_SCREEN;

  procedure CLEAR_LINE;

  procedure CLEAR_CURSOR_TO_EOL;

  procedure CLEAR_AND_HOME;
     -- Clears the screen and positions the cursor to the
     --     home position.



--Keyboard polling procedures:

  function GET_KEY return character;

  function KEYPRESSED return boolean;



  -- Controls for Reverse Video Printing:

  procedure SET_REVERSE(SWITCH : in SWITCHTYPE);

  procedure GET_REVERSE_STATUS(SWITCH : out SWITCHTYPE);
```

```
  -- Controls for Bold Face Printing:

  procedure SET_BOLD(SWITCH : in SWITCHTYPE);

  procedure GET_BOLD_STATUS(SWITCH : out SWITCHTYPE);


--Graphics character printing procedures:
      -- These procedures print one graphics character
      --   and leave the cursor on that character.
      -- If graphics mode is set when the procedure is called
      --   it will remain set. Otherwise graphics mode will
      --   be set ON and OFF to print the graphics character.

    procedure PUT_TOP_LT_CORNER;

    procedure PUT_TOP_RT_CORNER;

    procedure PUT_BOT_LT_CORNER;

    procedure PUT_BOT_RT_CORNER;

    procedure PUT_HORZ_BAR;

    procedure PUT_VERT_BAR;

    procedure PUT_CROSS;

    procedure PUT_TOP_TEE;

    procedure PUT_BOT_TEE;

    procedure PUT_LT_TEE;

    procedure PUT_RT_TEE;


-- Graphics Drawing Procedures:

  procedure DRAW_BOX(COLM,ROW : in integer);
    -- Parameters passed are the upper left corner
    --   of the box to be drawn centered on the screen.

  procedure DRAW_HORZ_LINE(LENGTH : in integer);

  procedure DRAW_VERT_LINE(LENGTH : in integer);
```

```
private

  --These procedures are called by other procedures in TERM_IO.
  --   They are not accessable to the programmer.

    procedure KEYPOLLER( KEY : out character);
     -- Implements the keyboard input method available on the
     --   terminal.


  --Graphics controlling procedures:

    procedure SET_GRAPHICS(SWITCH : in SWITCHTYPE);

    procedure GET_GRAPHICS_STATUS(SWITCH : out SWITCHTYPE);


end TERM_IO;
```

APPENDIX B

A PROGRAMMER'S GUIDE TO THE TERM_IO PACKAGE

This Appendix presents a programmer's guide to the use of the TERM_IO package. A programmer can use the TERM_IO package to control the screen format of a serial CRT screen that uses character control codes. To use the TERM_IO package, first ensure that the version of TERM_IO that is used is compatible with the terminal type in use. The control codes of various terminal types may be different.

The programmer has available in the TERM_IO package a set of convenient procedures that can be used to improve the screen output. These procedures include cursor control procedures, screen and line clearing procedures, keyboard polling procedures, printing status control procedures, and graphics procedures. The complete declaration of the package specification is contained in Appendix A. The programmer using the package should refer to the package declaration for the format of the TERM_IO procedure calls and the typing of the variables.

To use the package TERM_IO it must be included with each package body that calls a procedure in TERM_IO. This is done by including the statements

```
with TERM_IO;
use TERM_IO;
```

at the head of the package body that calls the procedures in TERM_IO. After this the procedures in TERM_IO can be called as if they were declared in the package.

Cursor Control Procedures

The cursor control procedures can be used to position the cursor to any position on the serial CRT screen. The programmer can use this feature to format the screen to presient menues or to display results. These procedures are

```
procedure MOVE_CURSOR_HOME;
procedure MOVE_CURSOR_UP(NUM : in integer);
procedure MOVE_CURSOR_DN(NUM : in integer);
procedure MOVE_CURSOR_RT(NUM :in integer);
procedure MOVE_CURSOR_LT(NUM : in integer);
procedure SET_CURSOR_POS(COLM,ROW : in integer);
```

The procedure MOVE_CURSOR_HOME moves the cursor to the upper left position on the screen. The procedures MOVE_CURSOR_UP to MOVE_CURSOR_LT allow the programmer to vary the amount of the cursor movement using the parameter NUM. This might be useful in applications such as drawing various size lines or positioning text based on the varying size of a figure. The procedure SET_CURSOR_POS(COLM,ROW : in integer) can be used to set the cursor position to any position on the terminal screen. A procedure call of SET_CURSOR_POS(0,0); is the equivalent of the call MOVE_CURSOR_HOME. In the original implementation the procedure GET_CURSOR_POS is not implemented. It always returns (0,0).

Screen and Line Clearing Procedures

The screen and line clearing porocedures can be used to erase all or part of the terminal screen. This is especially in an interactive situation where the programmer is changing only part of the display. These procedures are

```
procedure CLEAR_SCREEN;
procedure CLEAR_LINE;
procedure CLEAR_CURSOR_TO_EOL;
```

As the procedure names indicate, CLEAR_SCREEN erases the entire screen, CLEAR_LINE erases the entire line that the cursor is on, and CLEAR_CURSOR_TO_EOL erases the part of the current line to the right of the cursor including the cursor postion. None of the screen and line clearing procedures change the position of the cursor.

Keyboard Polling Procedures

The keyboard polling procedures can be used in interactive programming to get input from the program user. This input can be in the form of a one character answer such as a menu choice or hitting a key to signal that the user is ready to move to the next step. The keyboard polling functions are

```
function GET_KEY return character;
function KEYPRESSED return boolean;
```

The function GET_KEY is used to get a specific character response from the user. A call of

```
ANS := GET_KEY;
```

puts the user's answer in the variable ANS. One application of this feature is menu selection. The function KEYPRESSED returns true if the a key on the keyboard has been pressed. This can be used to allow the user to signal that he is ready to continue after reading instructions, for example. A simple waiting loop such as

```
while true loop
  if KEYPRESSED then exit; end if;
end loop;
```

waits for the user to hit a key on the keyboard.

Printing Status Control Procedures

The procedures for control of printing status allow the programmer to specify bold face or reverse video printing. This can be used to emphasize the headings displayed or the menu choices. These procedures are

```
-- Controls for Reverse Video Printing:
procedure SET_REVERSE(SWITCH : in SWITCHTYPE);
procedure GET_REVERSE_STATUS(SWITCH : out SWITCHTYPE);

-- Controls for Bold Face Printing:
procedure SET_BOLD(SWITCH : in SWITCHTYPE);
procedure GET_BOLD_STATUS(SWITCH : out SWITCHTYPE);
```

These procedures use the variable SWITCH of type (ON,OFF) to set the bold face or reverse video features. The procedures GET_REVERSE_STATUS and GET_BOLD_STATUS allow the programmer to check the status of these

features in the program. A call of

```
SET_BOLD(ON);
```

would set the output print on the terminal screen to boldface type until a call of

```
SET_BOLD(OFF);
```

restored the print type to normal.

Graphics Character Printing Procedures

The graphics character printing procedures allow the programmer to design simple shapes and graphs in Ada for the serial CRT terminal screen. These procedures are

```
procedure PUT_TOP_LT_CORNER;
procedure PUT_TOP_RT_CORNER;
procedure PUT_BOT_LT_CORNER;
procedure PUT_BOT_RT_CORNER;
procedure PUT_HORZ_BAR;
procedure PUT_VERT_BAR;
procedure PUT_CROSS;
procedure PUT_TOP_TEE;
procedure PUT_BOT_TEE;
procedure PUT_LT_TEE;
procedure PUT_RT_TEE;
```

These procedures each print a single graphics character on the screen at the current cursor location. A programmer can use these procedures to make a shape by positioning the cursor and printing each character at the desired screen location. The procedures leave the current cursor location at the location of the

graphics character. The programmer must reposition the cursor before each new character is printed.

Graphics Drawing Procedures

Several basic drawing procedures have been provided in addition to the character printing procedures. These procedures are

```
procedure DRAW_BOX(COLM,ROW : in integer);
procedure DRAW_HORZ_LINE(LENGTH : in integer);
procedure DRAW_VERT_LINE(LENGTH : in integer);
```

These procedures are easier to use to draw simple shapes than the character printing procedures. The procedure

```
DRAW_BOX(COLM,ROW : in integer);
```

draws a box centered on the screen with the upper left corner of the box at (COLM,ROW). The procedures DRAW_HORIZ_LINE and DRAW_VERT_LINE draw a line from the current cursor position of length LENGTH.

Private Procedures

The private procedures are not accessable to the programmer. These procedures are used by the other procedures in the TERM_IO package.

Sample Menu Maker

This procedure is an example of the use of the TERM_IO package. It shows how a programmer might use the functions of TERM_IO to write his own terminal interface procedures. It produces a simple menu and gets a menu selection from the user.

```
procedure MAKE_MENU(TITLE, OPT1, OPT2, OPT3,
                    OPT4, OPT5 : in string(30);
                    ANS : out character);

begin
  CLEAR_SCREEN;
  MOVE_CURSOR_HOME;
  DRAW_BOX(5,3);
  SET_CURSOR_POS(18,7);
  SET_BOLD(ON);
  put(TITLE);
  SET_BOLD(OFF);
  SET_CURSOR_POS(15,10);
  put("A. ");
  put(OPT1);
  SET_CURSOR_POS(15,12);
  put("B. ");
  put(OPT2);
  SET_CURSOR_POS(15,14);
  put("C. ");
  put(OPT3);
  SET_CURSOR_POS(15,16);
  put("D. ");
  put(OPT4);
  SET_CURSOR_POS(15,18);
  put("E. ");
  put(OPT5);
  SET_CURSOR_POS(5,23);
  put("Enter letter of choice, A - E:");
  while true loop
    ANS := GET_KEY;
    case ANS is
```

```
    when 'A'|'a'|'B'|'b'|'C'|'c'|'D'|'d'|'E'|'e' => exit;
    when others    => null;
   end case;
  end loop;
end;
```

This procedure constructs a standard menu on the screen. It draws a border, prints the menu title in boldface, presents five menu choices and waits for the user to respond. The menu title and options can be strings up to 30 characters long. The procedure MAKE_MENU shows how the programmer can use the TERM_IO package to format the screen output and control the movement of the cursor. When called in a program the procedure MAKE_MENU allows the programmer to set up the menu and get the user choice with as little code as possible. The result in the program is very clear and concise. A procedure call of

```
MAKE_MENU("Project Title",
        "Run Program",
        "Show Listing",
        "Set Options",
        "Get More Information",
        "Exit To System"
        ANS);
```

is all that is required to print the menu, wait for the user choice, and return the choice to the program.

APPENDIX C

TERM_IO PACKAGE BODY

```
--************************************************************
--
-- TITLE:        ADA TERMINAL INTERFACE
--
-- MODULE NAME:  TERM_IO BODY
-- DATE CREATED: 06 MAR 87
-- LAST MODIFIED: 15 MAY 87
--
-- AUTHOR:       LT Anthony J. Keough
--
-- DESCRIPTION:  This package body implements the
--    TERM_IO package for the VT - 100 terminal
--    using the ANSI control code sequences.
--
--************************************************************

with TEXT_IO;

use TEXT_IO;

package body TERM_IO is

-- Terminal parameters:

SCREEN_LENGTH : constant := 76;
SCREEN_HEIGHT : constant := 24;

-- ANSI Control code sequences

CLRSCR : constant string := (ASCII.ESC,'[','2','J');
HME : constant string := (ASCII.ESC,'[','f');
ONRVRS : constant string := (ASCII.ESC,'[','7','m');
OFFRVRS : constant string := (ASCII.ESC,'[','0','m');
UPCRSR : constant string := (ASCII.ESC,'[','1','A');
DNCRSR : constant string := (ASCII.ESC,'[','1','B');
RTCRSR : constant string := (ASCII.ESC,'[','1','C');
LTCRSR : constant string := (ASCII.ESC,'[','1','D');
CLREOL : constant string := (ASCII.ESC,'[','K');
CLRLNE : constant string := (ASCII.ESC,'[','2','K');
ONGRAF : constant string := (ASCII.ESC,'(','0');
```

```
OFFGRAF : constant string := (ASCII.ESC,'(','B');
ONBOLD : constant string := (ASCII.ESC,'[','1','m');
OFFBOLD : constant string := (ASCII.ESC,'[','0','m');

-- VT - 100 Terminal Graphics Characters:
--        These characters print as graphics characters
--        when the terminal is in the graphics mode.

UPLTCR : constant character := 'l';
UPRTCR : constant character := 'k';
DNLTCR : constant character := 'm';
DNRTCR : constant character := 'j';
HORZBR : constant character := 'q';
VERTBR : constant character := 'x';
CRSS : constant character := 'n';
UPTEE : constant character := 'w';
DNTEE : constant character := 'v';
LTTEE : constant character := 't';
RTTEE : constant character := 'u';


-- Terminal Status Variables:

REVERSE_STAT : SWITCHTYPE := OFF;
BOLD_STAT : SWITCHTYPE := OFF;
GRAF_STAT : SWITCHTYPE := OFF;

-- Other variables:

NUM : integer;


-----------------------------------------------------------------
-- Cursor Control Procedures:


procedure MOVE_CURSOR_HOME is

   -- Positions the cursor to the top left position.

begin
  put(HME);
end;


procedure MOVE_CURSOR_UP(NUM : in integer) is

begin
  for I in 1..NUM loop
    put(UPCRSR);
  end loop;
end;
```

```
procedure MOVE_CURSOR_DN(NUM : in integer) is

begin
  for I in 1..NUM loop
    put(DNCRSR);
  end loop;
end;



procedure MOVE_CURSOR_RT(NUM : in integer) is

begin
  for I in 1..NUM loop
    put(RTCRSR);
  end loop;
end;



procedure MOVE_CURSOR_LT(NUM : in integer) is

begin
  for I in 1..NUM loop
    put(LTCRSR);
  end loop;
end;



procedure MOVE_CURSOR_POS(COLM,ROW  : in integer) is

begin
  put(HME);
  for I in 1..COLM loop
    put(RTCRSR);
  end loop;
  for I in 1..ROW loop
    put(DNCRSR);
  end loop;
end;
```

```
procedure GET_CURSOR_POS(COLM,ROW : out integer) is

   -- This procedure is not implemented.
   --     To use this procedure a hardware specific
   --     call must be written for the procedure body.
   --     The procedure only returns (0,0).

begin
  COLM := 0;
  ROW  := 0;
end;

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

-- Screen and Line Clearing Procedures:

procedure CLEAR_SCREEN is

begin
  put(CLRSCR);
end;



procedure CLEAR_LINE is

begin
  put(CLRLNE);
end;



procedure CLEAR_CURSOR_TO_EOL is

begin
  put(CLREOL);
end;


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

-- Keyboard Input Procedures:


function GET_KEY return character is

    -- Gets a single character input form the user.

  KEY : character;

begin
  KEYPOLLER(KEY);
```

```
 return KEY;
end;


function KEYPRESSED return boolean is

    -- Returns true when the user has entered a key.


 KEY : character;

begin
 KEYPOLLER(KEY);
 return TRUE;
end;



procedure KEYPOLLER(KEY : out character) is

--This procedure can be modified to eliminate the
--  need for carriage returns by providing a
--  system specific keyboard polling routine.

begin
 get(KEY);
end KEYPOLLER;


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

-- Printing Status Control Procedures:


procedure SET_REVERSE(SWITCH : in SWITCHTYPE) is

begin
 if SWITCH = ON then
   REVERSE_STAT := ON;
   put(ONRVRS);
 elsif SWITCH = OFF then
   REVERSE_STAT := OFF;
   put(OFFRVRS);
 end if;
end;



procedure GET_REVERSE_STATUS(SWITCH : out SWITCHTYPE) is

begin
 SWITCH := REVERSE_STAT;
```

```
end;


procedure SET_BOLD(SWITCH : in SWITCHTYPE) is

begin
 if SWITCH = ON then
   BOLD_STAT := ON;
   put(ONBOLD);
 elsif SWITCH = OFF then
   BOLD_STAT := OFF;
   put(OFFBOLD);
 end if;
end;



procedure GET_BOLD_STATUS(SWITCH : out SWITCHTYPE) is

begin
 SWITCH := BOLD_STAT;
end;



procedure SET_GRAPHICS(SWITCH : in SWITCHTYPE) is

-- Private Procedure

begin
 if SWITCH = ON then
   GRAF_STAT := ON;
   put(ONGRAF);
 elsif SWITCH = OFF then
   GRAF_STAT := OFF;
   put(OFFGRAF);
 end if;
end;



procedure GET_GRAPHICS_STATUS(SWITCH : out SWITCHTYPE) is

-- Private Procedure

begin
 SWITCH := GRAF_STAT;
end;
```

```
-------------------------------------------------------

-- Graphics Printing Procedures


procedure PUT_TOP_LT_CORNER is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(UPLTCR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(UPLTCR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_TOP_RT_CORNER is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(UPRTCR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(UPRTCR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_BOT_LT_CORNER is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(DNLTCR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(DNLTCR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;
```

```
procedure PUT_BOT_RT_CORNER is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(DNRTCR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(DNRTCR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;



procedure PUT_HORZ_BAR is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(HORZBR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(HORZBR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_VERT_BAR is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(VERTBR);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(VERTBR);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;
```

```
procedure PUT_CROSS is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(CRSS);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(CRSS);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_TOP_TEE is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(UPTEE);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(UPTEE);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_BOT_TEE  is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(DNTEE);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(DNTEE);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;
```

```
procedure PUT_LT_TEE is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(LTTEE);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(LTTEE):
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;


procedure PUT_RT_TEE is

begin
  GET_GRAPHICS_STATUS(SWITCH);
  if SWITCH = ON then
    put(RTTEE);
    MOVE_CURSOR_LT(1);
  elsif SWITCH = OFF then
    SET_GRAPHICS(ON);
    put(RTTEE);
    MOVE_CURSOR_LT(1);
    SET_GRAPHICS(OFF);
  end if;
end;
```

```
procedure DRAW_BOX(COLM,ROW : in integer) is

   -- Draws a box centered on the screen with the upper
   --    left corner of the box at position (COLM, ROW)

begin
  CLEAR_SCREEN;
  SET_GRAPHICS(ON);
  MOVE_CURSOR_POS(COLM, ROW);
  put(UPLTCR);
  for I in COLM..(SCREEN_LENGTH - COLM - 2) loop
    put(HORZBR);
  end loop;
  put(UPRTCR);
  MOVE_CURSOR_DN(1);
  MOVE_CURSOR_LT(1);
  for I in ROW..(SCREEN_HEIGHT - ROW - 2) loop
    put(VERTBR);
    MOVE_CURSOR_DN(1);
    MOVE_CURSOR_LT(1);
  end loop;
  put(DNRTCR);
  MOVE_CURSOR_LT(2);
  for I in COLM..(SCREEN_LENGTH - COLM - 2) loop
    put(HORZBR);
    MOVE_CURSOR_LT(2);
  end loop;
  put(DNLTCR);
  MOVE_CURSOR_UP(1);
  MOVE_CURSOR_LT(1);
  for I in ROW..(SCREEN_HEIGHT - ROW - 2) loop
    put(VERTBR);
    MOVE_CURSOR_UP(1);
    MOVE_CURSOR_LT(1);
  end loop;
  SET_GRAPHICS(OFF);
end;
```

```
procedure DRAW_HORZ_LINE(LENGTH : in integer) is

begin
  SET_GRAPHICS(ON);
  put(LTTEE);
  for I in 2..LENGTH loop
    put(HORZBR);
  end loop;
  SET_GRAPHICS(OFF);
end;


procedure DRAW_VERT_LINE(LENGTH : in integer) is

begin
  SET_GRAPHICS(ON);
  put(DNTEE);
  MOVE_CURSOR_LT(1);
  MOVE_CURSOR_UP(1);
  for I in 2..LENGTH loop
    put(VERTBR);
    MOVE_CURSOR_LT(1);
    MOVE_CURSOR_UP(1);
  end loop;
  SET_GRAPHICS(OFF);
end;


end TERM_IO;
```

APPENDIX D

ANSI STANDARD AND VT - 52 TERMINAL CONTROL CODE SEQUENCES

Function	ANSI Terminal	VT - 52 Terminal
Cursor Up	ESC [Pn A	ESC A
Cursor Down	ESC [Pn B	ESC B
Cursor Right	ESC [Pn C	ESC C
Cursor Left	ESC [Pn D	ESC D
Home Cursor	ESC [H	ESC H
Position Cursor	ESC [Pl; Pc; H	ESC Pl; Pc
Reverse Video On	ESC [7 m	-
Reverse Video Off	ESC [0 m	-
Bold Face On	ESC [1 m	-
Bold Face Off	ESC [0 m	-
Graphics On	ESC (0	ESC F
Graphics Off	ESC (B	ESC G
Clear Screen	ESC [2 J	-
Clear Line	ESC [2 K	-
Clear To End of Line	ESC [K	ESC K

APPENDIX E

MODIFICATIONS TO THE ADAMEASURE USER INTERFACE

The package TERM_IO was applied to the AdaMeasure program to illustrate the use of the package and the benefits of the package in the areas of readibility and ease of use. The modified section of the AdaMeasure package MENU_DISPLAY is presented, followed by the original package. It should be apparent that using the TERM_IO package made the job of coding the user interface faster and easier.

Modified Package MENU DISPLAY

The TERM_IO package was used to modify the MENU_DISPLAY package. The MAKE_MENU procedure presented in Appendix B was used to modify the procedure MENU, illustrating how a standard menu procedure can be employed. The procedure INITIAL_MENU was modified using the TERM_IO functions directly for a 'custom design' interface.

```
with GENERAL_DATA, HALSTEAD_DISPLAY, INITIAL_DISPLAY, DISPLAY_SUPPORT,
    GLOBAL_PARSER, GLOBAL, TEXT_IO, TERM_IO;
use GENERAL_DATA, HALSTEAD_DISPLAY, INITIAL_DISPLAY, DISPLAY_SUPPORT,
   GLOBAL_PARSER, GLOBAL, TEXT_IO, TERM_IO;

package MENU_DISPLAY is
  procedure MENU;
  procedure INITIAL_MENU;
```

```
end MENU_DISPLAY;

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

package body MENU_DISPLAY is

 -- this procedure displays the metric selection menu from which the user
 -- can make the appropriate selection.
procedure MENU is

begin
 MAKE_MENU("METRIC SELECTION MENU",
        "'HALSTEAD' METRIC",
        "COMMENT AND NESTING METRIC",
        "'HENRY AND KAFURA METRIC",
        "EXIT TO MAIN MENU",
        "EXIT TO SYSTEM",
        ANS);
 case ANS is
    when 'A' => HALSTEAD;
    when 'B' => VIEW_GENERAL;
    when 'C' => VIEW_HENRY;
    when 'D' => DONE := TRUE;
    when 'E' => raise QUIT_TO_OS;
  end case;
 end loop;
end MENU;

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

 -- this procedure displays the main selection menu which allows the user
 -- to choose to parse a file, view previously gathered data, or quit to
 -- the operating system.
procedure INITIAL_MENU is

begin
 INTRODUCTION;
 CLEAR_SCREEN;
 MOVE_CURSOR_HOME;
 DRAW_BOX(6,4);
 SET_CURSOR_POS(20,7);
 SET_BOLD(ON);
 put("Main Selection Menu");
 SET_BOLD(OFF);
 SET_CURSOR_POS(12,9);
 put("1. ");
 put("Parse an input file");
 SET_CURSOR_POS(12,12);
 put("2. ");
 put("View Previously Gathered Data");
 SET_CURSOR_POS(12,15);
```

```
 put("3. ");
 put("Exit to operating System");
 SET_CURSOR_POS(6,22);
 put("Enter number of choice:");
 while true loop
  ANS := GET_KEY;
   case ANS is
    when '1','2','3' => exit;
    when others      => SET_CURSOR_POS(29,22);
                        CLEAR_CURSOR_TO_EOL;
   end case;
 end loop;
 case ANS is
   when '1' => RESET_PARAMETERS;
       INITIAL_SCREEN;
        MENU;
   when '2' => MENU;
   when '3' => raise QUIT_TO_OS;
 end case;
end INITIAL_MENU;

end MENU_DISPLAY;
```

Original Package MENU_DISPLAY

In the original package menu display the difficulties that the programmers had with the screen output are apparent. The programmers were required to specify the character for each screen loacation. This leads to the clumbsy repeated put and new_line calls. Even the spaces between the lines have to be printed in order to print the border. The asterisk symbol was used to put the border around the menu selections. The code is difficult to write and even harder to read.

```
with GENERAL_DATA, HALSTEAD_DISPLAY, INITIAL_DISPLAY, DISPLAY_SUPPORT,
   GLOBAL_PARSER, GLOBAL, TEXT_IO;
use GENERAL_DATA, HALSTEAD_DISPLAY, INITIAL_DISPLAY, DISPLAY_SUPPORT,
  GLOBAL_PARSER, GLOBAL, TEXT_IO;

package MENU_DISPLAY is
 procedure MENU;
 procedure INITIAL_MENU;
```

```
end MENU_DISPLAY;

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

package body MENU_DISPLAY is

  -- this procedure displays the metric selection menu from which the user
  --  can make the appropriate selection.
procedure MENU is
  DONE : boolean := FALSE;
begin
  while not DONE loop
    CLEARSCREEN;
    new_line;
    put("************************************************************");
    put("************************"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*                    METRIC SELECTION MENU       ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          HERE ARE THE INFORMATION CHOICES AVAILABLE");
    put(" TO YOU             *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*            Simply enter the number of your choice");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          1 - 'HALSTEAD' METRIC INFORMATION       ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          2 - COMMENT AND NESTING METRIC INFORMATION");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          3 - 'HENRY and KAFURA' METRIC INFORMATION ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          4 - EXIT TO MAIN MENU                 ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          5 - EXIT TO OPERATING SYSTEM          ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("************************************************************");
```

```
    put("************************"); new_line(2);
    put("Choice = ");
    get(ANSWER);
    get_line(DUMMY_FILE_NAME, LENGTH_OF_LINE);-- flush system input buffer
    new_line(2);
    case ANSWER is
      when '1' => HALSTEAD;
      when '2' => VIEW_GENERAL;
      when '3' => VIEW_HENRY;
      when '4' => DONE := TRUE;
      when '5' => raise QUIT_TO_OS;
      when others => null;
    end case;
  end loop;
end MENU;

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

  -- this procedure displays the main selection menu which allows the user
  -- to choose to parse a file, view previously gathered data, or quit to
  -- the operating system.
procedure INITIAL_MENU is
  DONE : boolean := FALSE;
begin
  INTRODUCTION;
  while not DONE loop
    CLEARSCREEN;
    new_line;
    put("*********************************************************");
    put("************************"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*                   MAIN SELECTION MENU          ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*           HERE ARE THE ACTION CHOICES AVAILABLE TO ");
    put("YOU               *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*             Simply enter the number of your choice");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          1 - PARSE AN INPUT FILE                ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
    put("*          2 - VIEW PREVIOUSLY GATHERED DATA        ");
    put("                *"); new_line;
    put("*                                        ");
    put("                *"); new_line;
```

```
    put("*          3 - EXIT TO OPERATING SYSTEM          ");
    put("                *"); new_line;
    put("*                                                ");
    put("                *"); new_line;
    put("*****************************************************");
    put("**********************"); new_line(2);
    put("Choice = ");
    get(ANSWER);
    get_line(DUMMY_FILE_NAME, LENGTH_OF_LINE);-- flush system input buffer
    new_line(2);
    case ANSWER is
      when '1' => RESET_PARAMETERS;
              INITIAL_SCREEN;
                MENU;
      when '2' => MENU;
      when '3' => raise QUIT_TO_OS;
      when others => null;
    end case;
  end loop;
end INITIAL_MENU;


end MENU_DISPLAY;
```

LIST OF REFERENCES

1. Neider, J. and Fairbanks, K., *AdaMeasure: An Ada Software Metric,* M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1987.
2. STARS Joint Service Team, *Preliminary System Specification,* Software Technology for Adaptable Reliable Systems, Software Engineering Environments, 30 January 1986.
3. Rosen, J.P., *On the Use of TEXT_IO on Interactive Terminals,* Ada Applications and Environments Conference, IEEE, October 1984, pp. 76-80.
4. Barnes, J.G.P., *Programming in Ada, 2nd ed.,* Addison-Wesley,Menlo Park, California, 1984.
5. ANSI / MIL-STD-1815A, *Military Standard,* Ada Programming Language, 22 January 1983.
6. Borland International, *Turbo Pascal Reference Manual,* Scotts Valley, California, June 1985.
7. Digital Equipment Corporation, *VT-100 User's Guide,* Maynard, Massachusetts, 1979.
8. Norton, Peter, *Programmer's Guide to the IBM PC,* Microsoft Press, Bellevue, Washington, 1985.
9. ANSI *Terminal Interface Standard,* New York, 1975.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Prof. Daniel L. Davis, Code 52Yy Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	5
5.	Center of Naval Analysis 2000 N. Beauregard Street Alexandria, VA 22311	1
6.	Dr. Ralph Wachter Office of Naval Research Arlington, VA 22217-5000	1
7.	Mr. Robert Westbrook CMDR, Code 33181 Naval Weapons Center China Lake, CA 93555	1
8.	Mr. Carl Hall Software Missile Branch Naval Weapons Center China Lake, CA 93555	1

9.	LT Anthony J. Keough Class 87080 Naval Submarine School Box 700, Code 20 Groton, CT 06349-5700	1
10.	Mr. Joel Trimble STARS Program Office OUSDR&E 1211 South Fern Street Arlington, VA 22202	1
11.	Mr. Harold Noffke AFWAL/AARM-3 Wright Patterson Air Force Base, OH 45433-6543	1
12.	Chief of Naval Operations Director of Information Stytems (OP-945) Navy Department Washington, DC 20350-2000	1

Thesis
K3863 Keough
c.1 TERM IO: an Ada terminal interface package.