Presented at the 1986 International Geoscience and Remote Sensing Symposium (IGARSS 86'), September 8-11, 1986, Zurich, Switzerland.

PEDITOR: A PORTABLE IMAGE PROCESSING SYSTEM

Gary Angelici, Sterling Software
NASA Ames Research Center
Ecosystem Science and Technology Branch
Moffett Field, California USA

Robert Slye
U.S. Department of Agriculture
National Agricultural Statistics Service
Washington, D.C. USA

Martin Ozga
University of California
Remote Sensing Research Program
Berkeley, California USA

Paul Ritter

ABSTRACT

The vast majority of image processing software systems can be used only on the type of computer system for which the software was written. The PEDITOR (Portable EDITOR) image processing system has been created specifically to be readily transferable from one type of computer system to another. While nearly identical in function and operation to its predecessor, EDITOR, PEDITOR employs additional techniques which greatly enhance its portability. After recounting the background of PEDITOR, this paper will describe the techniques developed in order to achieve portability. In addition, those experiences encountered in actually transferring PEDITOR to various types of computer systems will be shared.

Keywords: Image Processing, Portability, Pascal, Software, Programming Environment, Virtual Software Systems

1. INTRODUCTION

1.1 Background of PEDITOR

Since the early 1970's, the United States Department of Agriculture National Agricultural Statistics Service (USDA/NASS) has been involved in the processing of satellite data to generate periodic agricultural crop estimations. The software for this effort was developed initially on a Digital Equipment Corporation (DEC) PDP-10 computer running the TENEX operating system and served the basic purpose of configuring the data for processing on the ILLIAC IV supercomputer. This software package, called EDITOR, expanded over a decade to perform several additional functions and eventually evolved into a large image processing system of over 70 program modules and 100,000 lines of code. The software was written to support EDITOR's basic purpose, the generation of crop acreage estimates using satellite imagery, and performs numerous functions including: the digitization and plotting of boundary lines (e.g. agricultural field boundaries), the classification of multi-date multispectral data, block correlation and registration between two dates of imagery, and the tabulation of statistical data. Ozga and others (Ref. 1-2) have provided complete descriptions of the general structure and capabilities of the EDITOR system.

The USDA/NASS decided it would be necessary to move EDITOR to other computer systems if it were to continue to be useful. The EDITOR system, while functioning well, had been changed often. Also, it had been written using various programming languages, including assembler and others not readily available on other machines, and contained numerous machine dependencies inserted to improve efficiency and for other reasons. Thus, it was seen as useful to rewrite EDITOR in a more portable manner.

The PEDITOR (Portable EDITOR) image processing system was created to fulfill this requirement. Under the sponsorship of the USDA/NASS, programmers from the USDA/NASS, the NASA-Ames Research Center, and the Remote Sensing Research Program at the University of California participated in the development of PEDITOR. The purpose of the project was to retain the capabilities of the EDITOR system while, at the same time, greatly enhancing the portability of the software. The same program functions that exist in EDITOR, supporting the generation of crop acreage estimations from satellite data, were incorporated into the PEDITOR system. As will be shown, only the underlying structure was changed in order to allow the system to be ported to other computer systems. The Pascal programming language was selected because it is widely available and provides good features for structuring program statements and data. All of the PEDITOR modules and nearly all of the libraries were written in Pascal.

In order to fully test the portability of the new software system, two different types of computer systems running greatly differing operating systems were used as target machines. A DEC-20 computer running the TOPS-20 operating system and using a Pascal Compiler developed at Rutgers University was utilized for initial code development by the USDA/NASS programmers. The remaining programmers used a Motorola Corporation 68000-based Forward Technology FT-3000 supermicrocomputer running the UNIX-based XENIX operating system and using the Silicon Valley Software (SVS) Pascal compiler and the XENIX "C" compiler for their initial code development.

1.2 Requirements for Portability

For the PEDITOR project, portability was defined in terms of two basic criteria:

1) The vast majority of the PEDITOR program module and library source code must be able to be compiled without change on potentially disparate computer systems.

2) A mechanism must be available which would allow the creation of a simple procedure for compiling program modules and libraries on systems with varying compiler requirements.

2. PORTABILITY TECHNIQUES

A variety of techniques were conceived for the PEDITOR system in order to satisfy the two portability requirements. These techniques can be placed into two categories: system structure and preprocessing. While an indepth explanation of these techniques is offered in the PEDITOR manual (Ref. 3), the following description will serve as a thorough introduction.

2.1 System Structure

Most large applications software systems commonly consist of a number of program modules with a supporting collection of libraries. PEDITOR also has a similar structure, but due to the portability requirement, some different techniques were utilized. The most important of these techniques is the isolation of all machine dependent code into specific libraries.

Applications systems intended to run on a particular machine invariably contain machine-dependent code, such as operating system calls, in their modules and libraries. Such code in the PEDITOR system has been isolated to specific machine dependent libraries, so no machine dependent code exists in any of the PEDITOR modules or most of the libraries. Access to machine dependent features are made through one of the machine dependent libraries. All of the PEDITOR modules and libraries (except the machine dependent libraries) have been written in this manner, assuring that the vast majority of the PEDITOR source code is free of machine dependent code, thereby allowing their porting to other systems.

Since these machine dependent libraries contain system specific code, this portion of PEDITOR must be converted when transferring the software to another computer system. Using the library documentation and the existing code as implemented on another system, a programmer who is familiar with the new system can easily generate the necessary code for the new machine dependent library. Languages other than Pascal as well as machine-dependent extensions to Pascal can be used to implement the machine dependent libraries of PEDITOR. For example, the "C" language was used in the XENIX implementation of PEDITOR. The TOPS-20 implementation employed special features of the Pascal compiler allowing direct access to operating system calls. The proportion of this machine dependent code relative to the modules and the other libraries is very small. Of the over thirty libraries in the PEDITOR system, only two libraries contain machine dependent code.

The machine independent libraries perform a variety of functions including the reading and writing of data files, user prompting, and data manipulation. In addition, a general purpose library contains routines which provide a number of string handling and other generally usable capabilities. None of these libraries contain any machine dependencies and are able to be transferred to other computer systems without change.

2.2 Preprocessing

2.2.1 General. While the first requirement of portability, the ability to transfer most of the code without change, has been largely satisfied, a mechanism for compiling the code using any compiler must be developed.

By design, the source code for the modules and most libraries are kept completely free of machine dependencies, including compiler commands. Therefore, the source code could not possibly compile on any compiler without change. Hypothetically, each of the program modules and libraries could be edited to conform to the requirements of any particular compiler. This, however, could not be accomplished without making the code machine dependent once again and would require excessive manual labor. In the PEDITOR system, a separate program called the preprocessor has been developed to accommodate the various compiler requirements without changing the module and machine independent library source code.

The purpose of the machine dependent preprocessor is to convert machine independent source code into source code that is prepared for compilation using a particular compiler. Compiler specific syntax statements, such as compiler options, have been inserted into the preprocessor code itself by the PEDITOR system installer and, upon preprocessor execution, are written at the proper location in the compiler compatible file. Once the compiler compatible file has been created, the normal commands to compile and load programs for the host system can be used to generate an executable PEDITOR module.

2.2.2 Separate Compilation. The procedure described would be completely adequate if all compilers only allowed the direct inclusion of libraries during module compilation and did not offer the option of separate compilation of libraries. Because separate compilation of libraries is very pervasive, additional techniques, involving the preprocessor once again and requiring the implementation of a structure of ancillary files, had to be developed.

2.2.2.1 Preprocessor. Because any compiler requires different syntax for library versus module compilation, changes to the preprocessor were necessary. The code was changed to write different syntax to the compiler compatible file depending upon the type of source code which was input to the preprocessor, either module or library. The user also had to be prompted to identify the type of source code that was being entered.

The preprocessor is also the mechanism for processing a variety of ancillary files. Each of the files to be discussed in the next section are processed by the preprocessor in the effort to create compiler compatible files for both the direct file inclusion and the separate compilation methods of library compilation.

2.2.2.2 Ancillary Files. In Pascal, all procedures and functions must be defined within the program's

source code. When libraries are compiled separately, this requirement is met by using an "external" statement. There must be an external statement for each library routine to be used by the main program, letting the compiler know that these particular functions and procedures are defined elsewhere. In PEDITOR, the preprocessor must be able to support both source file inclusion and separately compiled libraries. To accomplish this in a manner that would be transparent to the module source codes, the module's include statement for a library refers to a "switch" file. This file contains another include statement, referring to either the library source code (for source code inclusion) or to a file containing the external statements for that library (for separate compilation). The type of file that is actually included into the compiler compatible file is set by the PEDITOR system installer according to the method of library compilation for that particular library. For each library (regardless of library compilation method), one such switch file exists. Also, for libraries to be separately compiled, a file containing the external declaration statements in the format required by the host compiler must be created. In a similar fashion, switch files are used for dealing with differences in the ways in which compilers handle global variables.

Even after being processed by the preprocessor and generating the proper compiler syntax, the format of the machine independent PEDITOR library files does not quite satisfy the declaration requirements of the compiler. Because the declarations for the other libraries that are required for library compilation are not in the PEDITOR library source code, a technique for incorporating those declarations was developed. Another type of ancillary file, the header file, contains all of the Pascal declaration statements (const, type, and var) along with include statements to the necessary PEDITOR machine dependent and independent source code files. By the use of an include at the beginning of the library source code file itself, this header file is inserted by the preprocessor at the top of the source code and written to the compiler compatible file. This include is not executed by the preprocessor if the direct file inclusion method of compilation is being performed. Once again, one of these ancillary header files is required for each library that is to be separately compiled.

The combination of PEDITOR system installer-selected ancillary files and educated processing by the preprocessor enable the creation of a compiler compatible file for both modules and libraries regardless of the method of compilation and without the need for programmer involvement.

3. EXPERIENCES AND CONCLUSION

3.1 Development Experiences

Throughout the development of PEDITOR concurrently on the XENIX and TOPS-20 systems, the portability of the system was tested. After a module was completed on one system, the exportable source code file was transferred to, compiled, and executed on the other system. While the vast majority of the code was successfully processed on the other system without error, a few problems did arise. As was discovered, many of the errors experienced during compilation and execution of the programs were caused by the lack of adherence either to the International Standards Organization (ISO) standard of Pascal programming or to generally accepted programming practices. But, because these problems were accommodated early in the writing of PEDITOR code, all subsequent code was relatively free of portability problems. Some of the problems encountered and their solutions implemented will be described briefly.

1) A limitation of 32,766 bytes as the maximum for one Pascal record in the SVS Pascal compiler did not exist in the TOPS-20 compiler. Any records containing more than the limit was subdivided into multiple records. Some changes to the source code itself was necessary to recognize the change in address of the newly re-located record structure members.

2) While the use of packed arrays of 32 bit values was allowed within a variant record in the TOPS-20 compiler, such a configuration was not allowed on the SVS Pascal compiler. Each reference to "packed array" for the full word data types had to be reduced to a simple "array" declaration.

3) Because most of the libraries on the TOPS-20 system were separately compiled, their ordering in the source code include section in the PEDITOR module was not important. When the source code was brought to the XENIX system, however, the unspecified ordering of the library files created an abundance of compilation errors stating that various routines did not exist. The routines did exist, of course, but the random ordering caused many of the routines to be called by the compiler prior to the compilation of the routines themselves. A reordering of library entries in the PEDITOR module source code solved the problem.

4) One entire category of errors relate to the effect of poor programming practices on different computer systems. For example, on the XENIX system, there are no undesirable consequences in neglecting to close a file opened for writing upon conclusion of the program. However, this dubious programming practice will result in the failure of the program to write the file at all on the TOPS-20 system. The common programmer error of neglecting to initialize data values and pointers also creates different run time errors on each of the systems. The TOPS-20 system will automatically initialize all global data values upon commencement of execution, whereas the global use of an uninitialized data value on the XENIX may cause unexpected program behavior. And by contrast, the XENIX system will automatically initialize all of its pointers while the TOPS-20 system will reliably abort the program stating that the pointers were not initialized.

5) One problem that was experienced early in the development of PEDITOR related to the handling of data types when performing input/output operations from a disk. If a number of bytes, halfwords, or full

32 bit words were read from a disk file, for example, combining or subdividing the data for use as a different type often resulted in an undefined output. While program interruption normally did not occur, the resultant values were incorrect. Care had to be exercised to be certain that if a data value was read in as a 32 bit value, for example, that any further processing treated the value as a 32 bit value and did not attempt to subdivide the value into its component bytes.

3.2 Additional System Porting

When the PEDITOR code was transferred from the XENIX machine to the TOPS-20 machine and vice versa during the development stage, all portability problems were overcome with relative ease. In the installation or attempted installation of the PEDITOR system on four other computer systems, some successes and some additional problems were encountered. The experiences of PEDITOR on the SUN microcomputer, the IBM PC, the VAX 11/780, and the IBM 30xx reveal more about the actual portability of PEDITOR.

Installation of the machine dependent portions of the PEDITOR system on a SUN 2/120 running Berkeley UNIX Version 4.2 was successful. Compilation of the primitive functions obtained from the XENIX implementation using the "C" compiler was accomplished without any code changes. The Berkeley Pascal Compiler, however, did show some inconsistencies with the SVS Pascal compiler. Comment statement delimiter closure and trailing blanks with strings are two areas in which the compiler implementation varied. Rather than change the preprocessor to accommodate the differences or remedially change all of the source code files, a compiler that was compatible with the SVS Pascal compiler was purchased for the SUN. Consequently, the source code compiled correctly without change, and the software is presently running smoothly.

Because of its popularity, an attempt to install PEDITOR on the IBM-PC/XT running the PC-DOS operating system was made. The inability of the compiler to accept the 32 bit declaration of "word" for use in array and "for" loop indices was a debilitating factor from an installation perspective. Because the "word" declaration is used throughout PEDITOR code for all integer values, all instances of "word" which are used as indices would have to be changed to "integer". Without an improved compiler for the PC, PEDITOR installation would be virtually impossible.

The VAX 11/780 running the VMS operating system was the most successful in accommodating the PEDITOR system. The primitive functions, obtained from the XENIX system, were written in "C" and compiled successfully. The preprocessor was adapted and compiled. The machine independent Pascal code was then compiled without change and over fifteen (15) modules have been implemented. Only the lack of user demand and programmer time prevent a full VAX implementation of the PEDITOR system.

As of this writing, the implementation of PEDITOR in the IBM mainframe environment under MVS/TSO is under way. While the machine dependent libraries were largely written using machine dependent extensions to Pascal, it was necessary in a few cases to write some routines in assembler. Due to differences in input-output implementation in the MVS environment from that in the TOPS-20 or UNIX environments, it was necessary to add a few new subroutines to the machine dependent library. Finally, a new portability problem occurred because IBM mainframes use the EBCDIC character set while other machines use ASCII. It was therefore necessary to change the read and write routines for binary data so that they could do the translation of embedded character data, thus allowing this data to be stored in ASCII so that files may be transferred between systems. Several modules have been ported onto the IBM 30xx, and a full implementation of PEDITOR is forthcoming.

3.3 Conclusions

An evaluation of the PEDITOR system as a portable software system will be offered on the basis of the portability requirements stipulated earlier. The isolation of the machine dependent portions of the software into specific libraries has largely satisfied the requirement that the majority of the software must be portable to other systems without change. Only 3 percent of the total source code of PEDITOR (the machine dependent libraries) must be adjusted in order for the software to be transferred to another computer system. Experience has shown, however, that other techniques, such as alternate compiler purchase or extensive preprocessor enhancement, may be necessary to enforce this level of portability for the machine independent portions of PEDITOR.

The ability to easily configure a compilation procedure for disparate computer systems has been provided by the preprocessor and the associated set of ancillary files. This technique has proved to be very straightforward in all computing environments. The programmer who wishes to augment or adjust the PEDITOR system finds the system structure very responsive to change and very easy to use.

PEDITOR has also demonstrated its portability in terms of its usefulness in agricultural remote sensing applications on disparate computer systems. For example, a large portion of the PEDITOR software was successfully used by the USDA/NASS on the DEC-20 system and by the California Department of Water Resources on the FT-3000 supermicrocomputer to generate crop acreage estimates from Landsat data in 1985.

Given its portability restrictions, the PEDITOR image processing system has a high probability of success in porting to many computer systems. Only with further attempts at porting the software system will the general portability of PEDITOR be known.

4. ACKNOWLEDGMENTS

The authors would like to thank those persons who contributed to the administrative support, design, programming, and testing of the PEDITOR system.

5. REFERENCES

1. Ozga M 1985, USDA/SRS Software for Landsat MSS-based Crop-Acreage Estimation, IGARSS, Amherst, Massachusetts 7-9 October 1985, 762-772.

2. Ray R et al 1975, EDITOR: An Interactive Interface to ILLIAC IV - ARPA Network Multispectral Image Processing Systems, Center For Advanced Computation Document No. 114, University of Illinois at Urbana-Champaign.

3. Angelici G et al 1986, The PEDITOR Image Processing System Manual, (presently in draft form).