A Datapro Report on

LPI

Language Processors, Inc. Multi-Language Family

Reprinted from

DATAPRO REPORTS ON UNIX SYSTEMS & SOFTWARE

Information that leads to action

datapro®

DATAPRO RESEARCH/A McGraw-Hill Company

McGraw Hill

DATAPRO RESEARCH □ 1805 Underwood Blvd. □ Delran, NJ 08075 USA □ (800) 328-2776 □ Telex 4761231 DPRO UI.

AUSTRALIA: McGraw-Hill Book Co., 4 Barcoo St., Roseville East, NSW 2069, Australia. Telephone (61) 2 406-4288. Telex 20849. MCGRAW AA.

CANADA: Datapro Research, 330 Progress Avenue, Scarborough, Ontario M1P 2Z5. Telephone (800) 668-9308. Telex 06525169 MCGRAWHILL TOR.

JAPAN: Nikkei Business Publications Subscription Sales Co., 1-14-6 Uchikanda, Chiyoda-ku, Tokyo 101, Japan. Telephone (81) 3 233-8081. Telex 29902 NIKKEIBPJ.

SINGAPORE: Datapro Research, Unit 05-03 Dapenso Building, 158 Cecil Street, Singapore 0106. Telephone (65) 222-5091. Telex 21912 DPRO RS.

SWITZERLAND: Datapro Research, SA, Case Postale 460, CH-1000 Lausanne 17, Switzerland. Telephone (41) 21 27 41 71. Telex 458196 DSSA CH.

UNITED KINGDOM: Datapro Research, McGraw-Hill House, Shoppenhangers Road, Maidenhead, Berkshire SL6 2QL England. Telephone (44) 628 773 277. Telex 848484 MCHILL G.

Language Processors, Inc. Multi-Language Family

SYNOPSIS

Language Processors, Inc. (LPI) is a major compiler vendor in the UNIX market. The company is set apart from competition because it offers several programming languages, making LPI a one-stop language source. LPI's Multi-Language family of compilers, consisting of Basic, C, Cobol, Fortran, Pascal, PL/1, and RPG II, is marketed through distribution agreements with computer manufacturers and distributors, in addition to direct catalog sales.

LPI's family of products centers around a modular software architecture, combining compiler subsystems, which are shared by LPI's different programming languages. For instance, LPI's cross-language calling can assemble a program using several different languages, so that existing software can be used without reprogramming. In addition, a common user interface provides the same system utility commands no matter what language is being used.

Products: Basic, C, Cobol, Fortran, Pascal, PL/1, and RPG II compilers.

Number of Installations: Between 10,000 and 15,000.

Current Versions: Basic 2.12, C 3.01, Cobol 5.65, Fortran 3.01, Pascal 2.11, PL/1 3.05, RPG II 3.0, CodeWatch 4.05 (LPI debugger).

UNIX Implementations Supported: AT&T System V Releases 2 and 3; 4.2 and 4.3 BSD; SCO Xenix; all major UNIX implementations.

Hardware Supported: Available on all Intel 386 systems, AT&T WE32XXX-based 3B systems, most Motorola 680X0-based systems, Motorola 88000 RISC-based systems, and Sun SPARC RISC-based systems.

Competition: Green Hills Software Inc., Micro Focus Ltd.

Standards Implemented: All LPI languages meet ANSI and X/Open standards.

Vendor: Language Processors, Inc. (LPI), 959 Concord Street, Framingham, Massachusetts 01701-4613. Telephone (508) 626-0006. Fax (508) 626-2221.

GSA Schedule: Yes.

Price: Compilers cost between $695 and $8,995, depending on target system size.

ANALYSIS

As open systems have emerged, the demand for software—including programming languages—available over a broad range of computer systems has also emerged. Multivendor networking has led to the need for portability among a range of systems. To avoid the extravagant cost of development for multiple systems, hardware manufacturers turn to third parties for software that conforms to industry portability standards.

Even though hardware vendors are reducing their involvement in software development, they still provide an important marketing vehicle for independent software houses such as LPI. Instead of direct marketing, LPI sells primarily through hardware manufacturers and distributors, saving both time and money by eliminating the overhead associated with individual, piecemeal sales. This move is advantageous for LPI, with the bulk of its income derived through these companies.

LPI's marketing strategy includes distribution agreements with over 20 domestic and 10 foreign computer manufacturers. In addition, it has agreements with multinational UNIX distributors, such as The Santa Cruz Operation and Interactive Systems. It also markets its compilers through direct catalog sales, a network of overseas distributors, and an affiliate in Japan called Nippon LPI (NLPI). Other overseas distributors cover France, West Germany, Brazil, Great Britain, Hong Kong, Thailand, Singapore, and Korea. (See Table 1 for more specific information on LPI's marketing base.)

In addition to its distribution network, LPI is targeting the government market as a major customer. Five of its compiler products were included in the recent AFCAC 251 Air Force contract awarded to AT&T. LPI and AT&T had been working together for two years to obtain the contract, calling for 20,000 AT&T 3B2/600 systems loaded with LPI compilers, totaling in excess of $1 billion over the next eight years. These systems constitute the largest federal procurement in history based on the UNIX operating system. The contract promises LPI visibility in the government market and will dramatically increase the installed user base of LPI software across the U.S.

COMPETITIVE POSITION

LPI is in a unique marketing position, offering seven different programming languages, where most compiler vendors offer only one. LPI's languages offer common programming tools, providing end users with the benefit of working with the same tools for all their language needs.

LPI competes against Green Hills Software, which also provides a family of compilers. LPI offers a wider variety of languages than Green Hills, however, targeting both scientific and business markets. Green Hills only competes in the scientific market, with offerings in Fortran, C, and Pascal languages. Another competitor is Micro Focus, offering a popular Cobol compiler; as with most other software houses, however, Micro Focus only offers one language.

Because, like LPI, distributors supply multiple languages, they constitute LPI's largest competitive base. Distributors market an assortment of products from single-compiler vendors, which is ideal for those users who need one language. But for those requiring several languages, LPI comes out ahead because it offers multiple compilers accessible through common tools and interfaces.

TABLE 1. LPI MARKETING BASE

Hardware Manufacturers	Software Developers	Software Distributors
Altos	SPSS	SCO
AT&T	Axis Corp.	Interactive
Apollo	Educational Testing (ETS)	UniSoft
Arix	Heuristima Corp	Nippon LPI
Convergent	Lawson Assoc	TopLog
Data General	Precision Systems Programming	Softway
Hewlett-Packard		Aval
Honeywell		
IBM		
MIPS		
NCR		
NEC		
Prime		
Tolerant		
Unisys		
Wang		

As a privately held company, LPI's management can exercise more control over company business than public companies. Because stockholders do not have to be considered, private companies have the flexibility to take risks, as with LPI's decision to acquire the competing Ryan McFarland Corporation (RM) in February 1989. This decision promises LPI more presence in the Cobol world, but first LPI will have to resuscitate Ryan McFarland, whieh some time ago lost its preeminent market position. RM will boost LPI's Cobol market share, bringing with it a large installed base of companies such as IBM, GE, Digital, and NCR. Even though LPI and RM service the same market, they will target different segments. In addition to new Cobol sales, RM will generate revenues through servicing customers already using RM-Cobol. LPI will continue to sell LPI-Cobol to those customers requiring additional languages.

Acquisitions are not the only strategy LPI has adopted to keep on top of the quickly changing computer market. In its continuing efforts to keep ahead of competition, LPI ships planned product updates to its customers every six months. Major revisions are provided faster, as standards evolve and fixes are required. This aggressive update policy ensures that customers always have the latest code.

LPI also tracks and supports new technologies aggressively. This policy recently gave LPI a competitive boost when it announced its support of the Motorola 88000 RISC and SUN SPARC architectures, making it the first language vendor to make a family of software development tools available for RISC-based architectures.

Of course, in order to be fully competitive in a rapidly standardizing market, LPI's products must adhere to industry standards, such as those mandated by ANSI and X/Open—and they do. LPI supports the efforts of both the Open Software Foundation (OSF) and UNIX International (UI), although it is not a member of either organization at this time, due to both organizations' current lack of involvement in compiler issues. Nevertheless, since these groups will influence the future of UNIX, it is very important for LPI to keep close watch over what they are doing. See Table 2 for language standards and extensions.

ADVANTAGES AND RESTRICTIONS

LPI gained a foothold in the federal sector through the recent AFCAC 251 contract. Because LPI is targeting the federal government, however, it is surprising that the Ada language has not been developed as a member of its language family. Ada is important to the federal sector and, in particular, to the armed forces in programming embedded systems for weapons control and other specialized functions.

Currently, LPI provides its customers more than just individual compilers. Modular architecture provides reliability because common components, shared among several languages, allow maintenance and support on fewer individual modules. Fewer modules, naturally, offer fewer places for a problem to occur. Troubleshooting is less complex, because technicians can test components rather than break down the entire system.

Cross-language calling is another advantage. This feature integrates existing programs written in any LPI language into new programs. It also means that new code does not have to be created and debugged if it exists elsewhere. For instance, a programmer writing a Pascal program wanting to manage data files, can use a data management portion of a Cobol program and integrate it into the Pascal program.

Although cross-language calling is attractive in theory, its value could be mitigated by an unpleasant reality: it can be a chore to track down old code for use in a new program. LPI has a viable answer to this problem. Its product, called Cocoon, can organize and track code, providing an object-oriented database that can store source code and assign traceable characteristics to it.

TABLE 2. LPI LANGUAGE COMPATIBILTIY

Language	Standards	Portability Extensions
Basic	ANSI X3.60-1978	CBasic Microsoft Basic
C	ANSI C X3J11	
Cobol	ANSI X3.23-1985 ANSI Cobol-68 ANSI Cobol-74	IBM/370 Cobol MF Level II Cobol RM/Cobol
Fortran	ANSI X3.9-1978 MIL-STD-1753	VAX-Fortran Fortran-66
Pascal	ANSI/IEEE 770 X3.97-1983 ISO Level 0	
PL/1	ANSI PL/1 X3.74-1981	IBM-PL/1 VAX-PL/1

LPI's compilers are true compilers, not interpreters. This fact is especially important to users who develop large programs. Interpreters are intrinsically slower; they translate and execute source code line by line. If a program is used multiple times, the sequence has to be repeated every time because the compilation never produces an object file. True compilers go through fewer compilation steps; they compile only once, producing object code that is used whenever the program is run again.

USER EXPERIENCE

Datapro conducted telephone interviews with two LPI users in March 1989. These users, both located in the United States, were chosen from a list of customer references supplied by LPI.

Site One: We spoke to an analyst working for a software development house in the Midwest. His firm develops manufacturing applications based on UNIX System V Versions 2 and 3. He has dealt with LPI for five years and is satisfied with the company and its products. His company provides a manufacturing application based on LPI-Cobol, with outside C subroutines. He said that he has had no problems interfacing the two languages.

The only problems this company has experienced are with software revisions from LPI. Serious bugs have shown up after installation of fixes for minor bugs. Additional corrections, however, always resolve the second set of bugs. The problem of fixes creating bugs may seem major, but the user stressed that this type of problem commonly happens in software development, regardless of which vendor's software is used. The user stated that LPI has always fixed bugs in a timely manner—the longest time it has ever taken LPI to fix a bug is one month. This analyst recommends LPI to others, citing excellent support and ease of use as major benefits. He stated, "We're probably LPI's best customers."

Site Two: We interviewed a system administrator of a major corporation located in the Great Lakes region. This user is overseeing a PL/1 conversion from a Honeywell mainframe running the Multix operating system to Apollo desktop units running AT&T UNIX System V. The department is presently halfway through the code conversion. When asked why he chose LPI, he stated that the company was the only vendor that supplied PL/1 for Apollo systems. Since LPI offers only the standard PL/1, this company must rewrite the code where Multix extensions were originally used.

The system administrator stated that the programmers have had to call LPI several times with questions and have always received satisfactory answers. Usually, though, the programmers are able to find answers in the documentation. When the department started the conversion, features of Honeywell's PL/1 and LPI's PL/1 had to be compared. The user said that LPI's documentation clearly laid out the features, which this user found extremely helpful.

CHARACTERISTICS

ARCHITECTURE

The LPI family of language standards includes Basic, C, Cobol, Fortran, Pascal, PL/1, and RPG II. Development tools supporting these languages include LPI's CodeWatch, a source-level debugger; CoEdit, a language editor; and Cocoon, a development environment organization tool.

Each LPI compiler employs a component architecture made up of five subsystems—the front end, optimizer, code generator, run-time library, and source-level debugger. Using this common architecture, subsystem components can be shared between languages. For instance, programs written in both Basic and Pascal can be debugged with a common debugger. This architecture also allows developers to move applications between hardware platforms by recompiling.

The front end (i.e., the compiler itself), which contains the syntax and semantics analysis for each language, differs among the various compilers. The remaining components, common to all LPI compilers, are explained in the following sections. A separate technical description will be supplied for each compiler in the product line. Pricing and support information for all products is grouped together after the last technical description. Figure 1 illustrates LPI's product relationships.

User Interface

The user interface is command-line driven. Users, however, can implement interactive windows and pop-up menus to aid in program development for use with bit-mapped terminals. The commands for all utilities are the same no matter what language is being used; only the commands inherent in the specific language are different. (See the section on CoEdit Editor below for more information about these facilities.)

Level of Machine Code Generated

LPI compilers are true compilers, translating source code first into an intermediate language and then into machine code. LPI's intermediate language is the proprietary LPSI, which retains many of the characteristics of the source code and is optimized to eliminate inefficient operations or redundancies. When LPSI is translated into machine code, it is stored. Subsequent executions in machine code are executed at the speed of the machine's hardware. There is no translation after the program has been compiled. The product line has no artificial limitations, such as the number of source lines compiled, symbol table size, file size, or data size (i.e., no 64K-byte limitation).

LPSI is the one common point among LPI's seven diverse computer languages (front ends) and many different code generators (back ends), such as industry-standard processors and a range of proprietary processors or architectures. This connecting point to back ends is used as the interface for manufacturers adopting or developing 4GL and CASE products.

Language Processors, Inc. Multi-Language Family

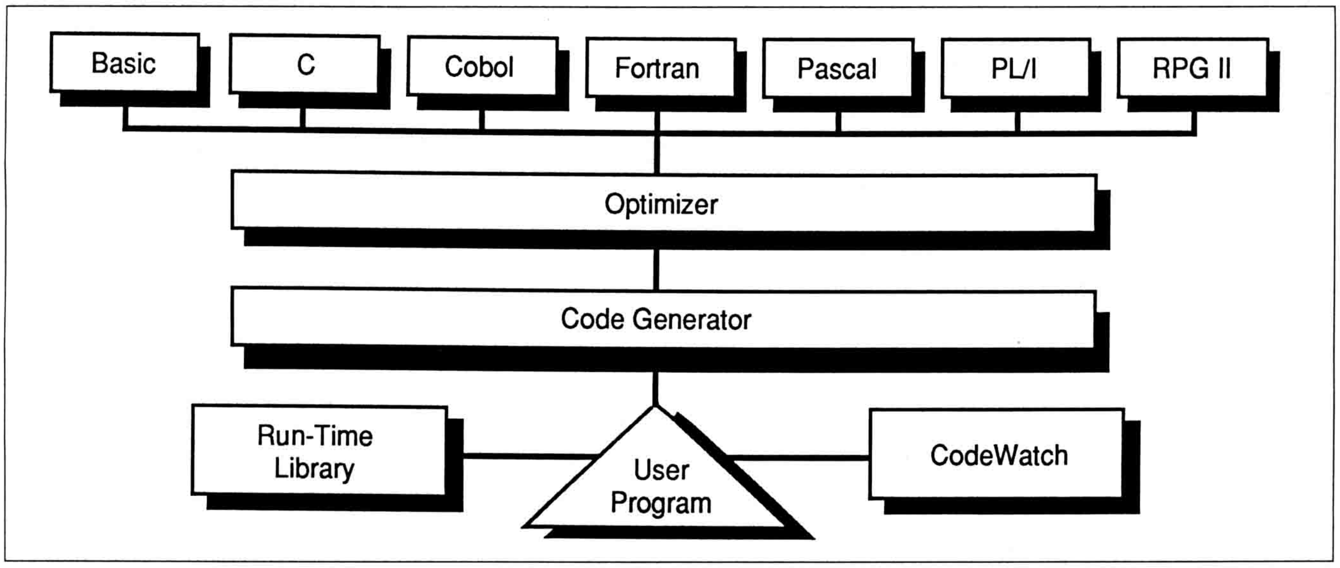


Figure 1. LPI's component architecture is the technology underlying each compiler. Its five subsystems consist of the front end (programming language), optimizer, code generator, run-time library, and CodeWatch debugger. The front end contains the syntax for each language and differs among all compilers. All the other subsystems are common to all compilers.

Code Generator

The modularity of LPI's component architecture permits movement of the language family to different processors by creating a new code generator and replacing approximately a quarter of the run-time library. A new code generator can be created using LPI's Code Generator Generator, which produces the major portion of a new code generator by describing the characteristics of the processor for which the generator is being created. The entire language family can then be ported to the new system. LPI provides code generators for the following hardware architectures:

- Motorola 680X0 microprocessor
- Motorola 88000 RISC processor
- Intel 386 microprocessor
- Sun Microsystems SPARC RISC processor
- AT&T WE32XXX microprocessor
- National Semiconductor NS32XXX microprocessor

Optimizing Facilities

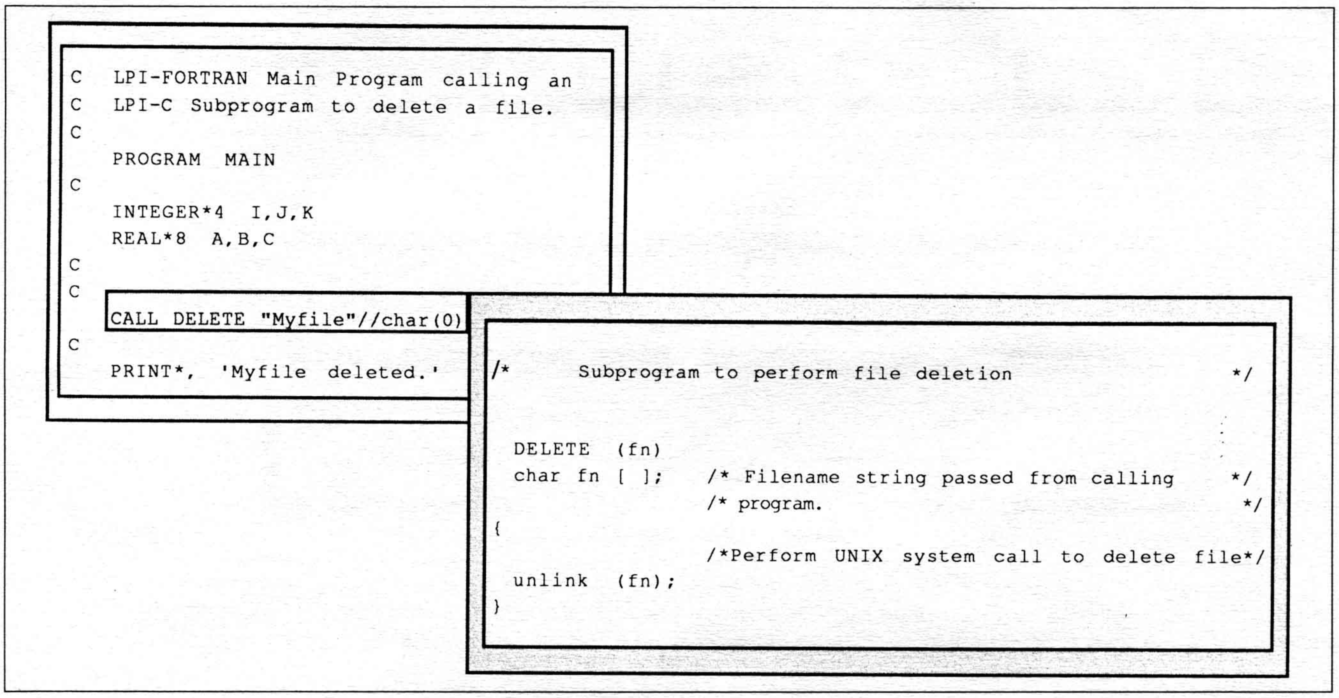
All LPI compilers use a common optimizer. Optimization facilities are both global, across entire program units, and local—within individual statements or expressions. Three levels of optimization are available. No optimization is useful for system checking, program debugging, and ensuring the fastest possible compilation time. The intermediate level of optimization eliminates redundant computations and optimizes branches and logical expressions. The highest level of optimization generates the most efficient run-time code. In addition, machine-dependent optimizations can be performed to make the most efficient use of the processor hardware.

Language-specific optimization is in the front end (individual languages), and peephole optimization is in the back end (code generators). Specific features include constant folding, loop induction, dead code elimination, and common subexpression elimination.

Cross-Language Calling

Through cross-language calling, programmers can integrate new languages into a program while maintaining existing subroutines. Thus, a programmer is not restricted to a single language when writing application programs but can use the language best suited to the given programming task. For instance, an application's numerical processing tasks can be written in Fortran and then accessed by a main program written in PL/1.

Cross-language calling also allows programmers to integrate portions of existing programs written in any LPI language into new programs so that they do not have to create and debug new code. LPI's debugger recognizes the language being used and communicates with the programmer in the conventions of that language.



```
C   LPI-FORTRAN Main Program calling an
C   LPI-C Subprogram to delete a file.
C
    PROGRAM  MAIN
C
    INTEGER*4  I,J,K
    REAL*8  A,B,C
C
C
    CALL DELETE "Myfile"//char(0)
C
    PRINT*,  'Myfile  deleted.'
```

```
/*      Subprogram to perform file deletion                    */


  DELETE  (fn)
  char fn  [ ];     /* Filename string passed from calling     */
                    /* program.                                 */
{
                    /*Perform UNIX system call to delete file*/
  unlink  (fn);
}
```

Figure 2. This program demonstrates the essence of cross-language calling—developers target the best language to the given programming task. This subroutine can be called into any other program written in an LPI language. In this example, a Fortran program calls a C file deletion routine.

This feature can also be extended to native C programs. LPI compilers produce object code readable by the system linker and loader, allowing integration into other applications, such as DBMS or graphics packages and into applications developed with LPI compilers. See Figure 2 for an example of cross-language calling.

Run-Time Library

An extensive run-time library is common to all of the compilers. The run-time library is a set of routines that save the effort required to do many of the system's standard internal functions. The run-time library contains routines that include the following functions:

- Math routines
- General utility library, e.g., data type conversion
- Commercial instruction set simulation
- Language-dependent routines
- Operating system interface routines
- File access interface routines
- Common multikey, indexed sequential file-handling routines

Error Definition and Handling

Full support to aid users in the diagnosis and correction of programming errors is provided. Error messages specify which conditions have been encountered and where in the source program they were found. Messages also identify the severity of the error, specific information about the error, and a probable solution.

Editing and Debugging Facilities

LPI offers an editing facility, *CoEdit*, and a debugging facility, *CodeWatch*. CoEdit and CodeWatch provide common facilities for programmers to write, test, and debug programs using the conventions and symbols of different source languages. Separate descriptions of CoEdit and CodeWatch are explained in the following sections.

CoEdit Editor

CoEdit is a language-sensitive editor, integrated with LPI compilers and diagnostics; it works with both conventional and bit-mapped terminals. Commands are systematically organized into logical mnemonic groups. In addition to programming features, CoEdit provides a macro programming environment, with a compiler and debugger for the macros. Features include block, locate and replace, and tab functions.

Editing Facilities: Text can be moved between files; files matching a certain pattern can be loaded sequentially or simultaneously. Windows can be tiled, overlapped, expanded, hidden, or marked for read-only. The windows have line continuation and overflow indicators on the window borders. Pop-up menus locate commands without the user's having to leave the text or look through the manual. Menus can be turned off once commands are learned. Context-sensitive online help is available at most prompts; directory listings are also available at file prompts.

The keyboard is reconfigurable, allowing the assignment of command keys to be changed. CoEdit saves up to 36 sequential backup source code files. It can perform automatic background saving, storing work in special files to prevent loss in the event of system or power failures. CoEdit's undo feature remembers over 32,000 deletions, allowing access to any one of them.

Error Handling: CoEdit has a built-in command to compile source code. When compilation is complete, CoEdit highlights any lines in the source file that were flagged by the compiler as containing errors and places the cursor at the first error. The corresponding error message from the compiler appears on the border of the window, allowing errors to be fixed when they appear. This facility eliminates the need for error listings or viewing error output. When the error is fixed, the error status indicator for that line is turned off and moves to the next error.

Syntax Checking Facilities: CoEdit also provides precompilation syntax checking for properly balanced parentheses, brackets, or braces. When entering a closing delimiter, CoEdit flashes the cursor at the corresponding opening delimiter.

Programming Tools: Several tools are available within CoEdit. An expression evaluator performs computations and shows the results in binary, octal, decimal, and hexadecimal format. Additionally, a table displaying octal and hexadecimal equivalents of ASCII or EBCDIC characters can be displayed online. ASCII or EBCDIC values of specified characters can also be displayed within text.

An on-screen file display command lists those files that can be accessed from a given directory, including date and time stamps. Programs or system commands can be executed from within the editor or by exiting to the UNIX shell. Output from any command can be inserted directly into text.

Macro Capabilities: CoEdit's macro language is similar to C and Basic and includes its own macro compiler and source-level debugger. An unlimited number of macros can be created and recorded directly from the keyboard for reuse of a key sequence. Macros can also be entered in source form and compiled. If there is a mistake in a macro, the source-level macro debugger can determine the problem.

Locate and Replace: Editing functions include Locate and Replace commands with an option that finds all occurrences of a pattern. All lines containing the pattern are gathered in a subwindow, which can then be edited as a group. When these lines are returned to the parent window, the changes are automatically replaced. Locate and Replace commands can find or replace regular expressions and are optionally case sensitive. Locate or Replace can be done in a forward or backward direction or within a block definition or line range. Optional prompts are available to verify each replacement.

Blocks: Each window can have its own block defined. Blocks can be copied, moved, or deleted within the file and copied or moved within or between buffers. The Block Filter command extends CoEdit functions with any user-defined or system filter function. CoEdit can automatically take the contents of a block and replace them with the output from the filter. This operation can be undone because the original block is saved in the delete buffers before the filter operation is performed.

Tabs: The Tab feature can dynamically change tab widths. Available types are soft tabs, hard tabs, and fixed tabs. Fixed tabs allow manual tab sets to be made at any column position.

CodeWatch Debugger

CodeWatch is LPI's interactive source-level debugger for its Basic, C, Cobol, Fortran, PL/1, and Pascal compilers. CodeWatch works on actual source code without using an interpreted intermediate language. It allows programmers to interact in the conventions and symbols of the source language, as well as track program variables in the source language.

Each CodeWatch command has an alternate abbreviation; for example, the abbreviation ARG can be entered instead of ARGUMENTS. Also, on-line help provides information on the use of each CodeWatch command for reference during debugging sessions. See Table 3 for CodeWatch commands.

Language Processors, Inc. Multi-Language Family

TABLE 3. CODEWATCH COMMANDS

ARGUMENTS	Prints the arguments of a specified procedure environment.	MACRO	Defines and names a macro.
BREAKPOINT	Specifies the point at which program execution is to be suspended for debugger actions.	NBREAKPOINT	Removes one or more breakpoints.
COMMANDLINE	Prints the command line argument with which the user program is invoked.	NLOG	Stops logging of debugger commands to a file.
CONTINUE	Begins or continues program execution.	NMACRO	Removes one or more macro definitions.
DSTEP	Sets the default stepping mode, either OVER, IN, or OUT, and sets the action list for the STEP command.	NTRACE	Disables entry or statement tracing.
ENVIRONMENT	Sets the evaluation environment.	POINT	Sets the source file display pointer within the current source file.
EVALUATE	Evaluates and prints the resultant value of expressions in the source language.	PRINT	Prints a specified number of lines from the current source file.
FIND	Locates and prints a specified string in the source code.	QUIT	Terminates the debugging session.
GOTO	Moves the execution pointer to a specified statement.	READ	Reads and executes debugger commands from a specified file.
HELP	Lists debugger commands and operations.	RELOAD	Reinitializes the user program.
LBREAKPOINT	Lists information on one or more breakpoints.	RETURN	Transfers the execution pointer to the exit point of the current subroutine, and returns a value, if necessary.
LENVIRONMENT	Lists the current evaluation environment.	SOURCE	Changes the current source file to be displayed.
LET	Assigns the value of an expression to a variable.	STACK	Displays various information on a specified number of stack frames.
LMACRO	Lists one or more user macro definitions.	STEP	Executes a specified number of statements.
LOG	Logs debugger commands to a file.	TRACE ENTRY	Displays subroutine, block entry, and exit point information during program execution.
LRETURN	Prints the return value of a function.	TRACE STATEMENT	Displays source info during program execution.
LSOURCE	Prints the name of the current source file being displayed.	TYPE	Prints the resultant type of an expression.
LSTEP	Lists the current stepping mode, IN or OVER, and the default action list.	WHERE	Prints the current execution point or the location of a specified statement.
LTRACE	Prints the current tracing mode.	*!command*	Invokes the command interpreter to perform operating system commands.

Program Execution: Execution of a program can be started or suspended and resumed at user-specified points. Execution can also be transferred from one point to another, including a subroutine exit point. A return value can be set at this exit point.

Breakpointing: This capability allows the developer to suspend program execution at any source statement or at the entry or exit point of any subroutine. As each breakpoint is encountered, a series of commands and checks can be performed automatically. Conditional breakpointing permits automatic breaks under specified conditions. Skip counts provide flexibility by allowing execution to advance over a given number of breakpoints for rapid access to errors.

Stepping: With this facility, execution of one or more source statements can be made at one time. Developers can step one statement at a time into, out of, or over a called subroutine. This facility helps determine the effects of certain results later in the program.

Tracing: This enables information about user-specified statements and subroutines to be reported as the program executes, thus allowing the isolation of errors to a particular program unit.

Language Processors, Inc.
Multi-Language Family

Action Lists: Several debugger commands can be grouped into a unit and executed at user-specified breakpoints, steps, or tracepoints. These action lists allow developers to design and implement commands for specific debugging requirements.

Macros. Macros allow users to create customized debugger commands interactively or read them from a file. Once defined, macros become a single debugger command that can be used as shorthand for debugging a program, ensuring consistency of debugging efforts throughout a debugging session.

Development Control

LPI provides an object-oriented software development tool, called *Cocoon*, which provides sets of frameworks specifying procedures and guidelines that govern a software development project. Cocoon integrates and organizes user programs, system commands, and procedures. It provides the source, version, and product status control; manages the configuration of the environment; and maintains history records. It includes an interactive window and menu interface system, as well as an object-oriented database.

Cocoon conforms to the development environment in place. For example, using Cocoon's object orientation, text, source code, defects, executable code, or other types of data can be described. This data is then assigned characteristics and recorded.

Two kinds of frameworks, i.e., sets of classes, objects, and methods that implement a desired function, can be used with Cocoon:

- LPI-designed frameworks, which can be used and/or modified, including Online Documentation, Software Development, and Documentation Control frameworks
- User-created object-oriented frameworks (such as those for prototyping a proposed application)

Either of these frameworks can be integrated into existing environments within the development process.

Online Documentation

Online documentation can be created with the general Cocoon framework, ranging from *man* pages in the UNIX environment to a linked hypertext environment where users move around online documentation as needed. This framework allows online information to be ported to every system on which the application runs. The Online Documentation framework allows application developers to semiautomatically create, format, and link frames of information into an online documentation system. The application end user can select a topic by highlighting it and viewing the corresponding frame. Each frame then offers a choice of what to see next. The reader can select and learn about other terms in the text through a pop-up window.

Software Development

Cocoon's Software Development framework manages software development functions, including configuration management and source and version control. The configuration management feature defines and manages the software modules used to build a product. The source and version control feature maintains different versions of source code, allowing re-creation of older versions and variants of a product.

Documentation Control

The Documentation Control framework allows streamlining of repetitive documentation efforts. With this system, writers define which changes need to be made to which books and enter those changes one time. Documentation is then updated automatically on all indicated books at once.

Customization

Any Cocoon framework can be customized as needs change. Cocoon is fully programmable and allows the integration of user programs, system commands, and procedures into a consistent whole through an object-oriented database and a conventional programming environment.

Developers can create procedures geared toward specific or unique software development activities by defining certain object classes and specified operations to be performed on those items. Basic object classes include character strings, numbers, sets of objects, and references to other objects. In addition, *attributes* convey additional information about the objects. Programmers can define any number of object classes derived from the basic class.

LPI-Basic Technical Description

LPI-Basic is an implementation of the American National Standards Institute (ANSI) Minimal Basic

Language Processors, Inc.
Multi-Language Family

TABLE 4. LPI-C FUNCTIONALITY

	Statements	
break	continue	do-while
for	goto	if
if-else	null	return
switch	while	
	Data Types	
*(pointer to)	char	double
enum	float	int
long	short	unsigned char
unsigned int	unsigned long	unsigned short
	Special Characters	
'\r'(carriage return)	'\n'(newline)	'\f'(form feed)
'\t'(tab)	'\b'(backspace)	'\\'(backslash)
'\ddd'(octal constant)	'\"(single quote)	
	Reserved Words	
auto	break	case
char	continue	default
do	double	else
entry	enum	extern
float	for	goto
if	int	long
register	return	short
sizeof	static	struct
switch	typedef	union
unsigned	void	while
	Preprocessor Commands	
#define	#else	#endif
#if	#ifdef	#ifndef
#include	#line	#undef

X3.60-1978 standard. It allows conversion of existing applications from other Basic compilers and applications developed in Microsoft's MBasic and Digital Research. Inc.'s CBasic. It takes advantage of other LPI functions. including cross-language calling. the CodeWatch debugger. and listing options. When errors are found. a complete sentence error message is supplied that includes the exact line number of the error and indicates probable solutions.

LPI-C Technical Description

LPI-C is an implementation of the industry-standard C. as defined by Kernighan and Ritchie in *The C Programming Language*. LPI will also release ANSI C in June 1989. LPI-C incorporates extensions to enhance compatibility with recent C implementations. including full support of enumerated data types. structure assignment. void data type. the "defined" preprocessor. and the "_LINE_" and "_FILE_" predefined preprocessor identifiers.

An integrated preprocessor compiles programs in one step. eliminating the need to run several procedures. LPI-C is compatible with *cc* command line options and existing *make* files. Users can invoke their program using either C options or *cc* compiler options. Error messages identify the location of an error by placing a pointer at the exact location within the line. See Table 4 for specific LPI-C functions.

LPI-Cobol Technical Description

LPI-Cobol fully implements the ANSI Cobol X3.23-1985 programming language. It is compliant with X/Open. Language extensions support the porting of RM/Cobol. Micro Focus Level II Cobol. IBM/370 Cobol. Cobol-74. and Cobol-68 applications. Converted applications can be ported to several hardware platforms. including the Motorola 680X0 series. National Semiconductor NS32XXX. AT&T WE32XXX. and Intel 386.

ANSI modules implemented at the high level include Nucleus. Sequential I/O. Relative I/O. Indexed I/O. Sort/Merge. Inter-Program Communication. Source Text Manipulation. and. optionally. Segmentation. Federal Information Processing Standard (FIPS) flagging is offered. allowing users to note extensions beyond a given FIPS level.

Since LPI-Cobol is a 32-bit compiler. there are no artificial limits on program size due to hardware architecture. Program size and the amount of data are restricted only by the amount of memory available on the system.

The run-time system provides a multikey indexed sequential file handler and Sort/Merge routine for applications development. The Copy statement provides file sharing among Cobol programs. permitting modular programming and program abstraction. An ISAM database system. using C-ISAM revision 3. is supported. permitting users to look up records in Cobol by using primary or alternate keys or by relative access. Record locking is also supported. providing data integrity. Interfaces to Oracle. Unify. and Informix relational database management systems are possible through C routines.

LPI-Fortran Technical Description

LPI-Fortran is a full implementation of the ANSI Fortran X3.9-1978 (Fortran-77) and MIL-STD-1753 and is X/Open compliant. It provides extensions to

Language Processors, Inc.
Multi-Language Family

TABLE 5. LPI-FORTRAN EXTENSIONS

- 32-character name lengths.
- Data initialization in type statement.
- Equivalence of noncharacter with character entities.
- Conditionally compiled lines.
- One-trip DO loop compiler option.
- IMPLICIT NONE statement.
- Logical data types in arithmetic expressions.
- ! as a comment indicator.
- Retrieval of command line arguments.
- Hollerith constants.
- DO WHILE loops.
- Octal and hexadecimal constants.
- Recursion support for subprograms.
- INCLUDE external files statement.
- Bit intrinsic operations.
- Tab formatted source lines.
- Ampersand (&) continuation lines.
- # carriage control character.
- No column 72 limit.
- Lowercase characters.
- Case insensitivity/sensitivity.
- Nonalphabetic characters in names.
- Extended logical, integer data types.
- Data initialization flexibility.
- ".." or '.' character constants.
- Save-all compiler option for data.
- Common blocks treated as save.
- Subscripts in equivalence statement.

help transport existing applications developed on Digital and IBM systems to UNIX systems.

Enhancements include recursive subroutines and bit intrinsic functions. Recursive subroutines provide flexibility in that a program subroutine can reference itself; bit intrinsic functions allow users to manipulate the number of bits representing internal data types. See Table 5 for LPI-Fortran extensions.

LPI-Pascal Technical Description

LPI-Pascal is an implementation of the ANSI/IEEE Pascal 770X3.97-1983 standard, and conforms to the ISO/7185 Level 0 standard. It allows conversion of existing applications from other Pascal dialects by recompiling. Separate compilation of Pascal program units is supported, facilitating the development and maintenance of structured program units, and allowing implementation of minor modifications. In addition, external subroutines can be called. Error messages provide the exact line number of the error and indicate possible fixes. Four classes of error messages are provided, all with different warning levels. See Table 6 for LPI-Pascal extensions.

LPI-PL/1 Technical Description

LPI-PL/1 is a full implementation of the ANSI PL/1 X3.74-1981 General Purpose Subset. Language extensions afford compatibility with PL/1 dialects for systems such as Digital VAX minicomputers and IBM mainframes. Applications using LPI-PL/1 can interface with existing system libraries, providing access to graphics and statistical analysis capabilities.

LPI-RPG II Technical Description

LPI-RPG II is an implementation of IBM System/34 and System/36 RPG II. It includes an RPG II compiler, OCL processor, and utilities to help transport RPG II programs from IBM systems to UNIX and Xenix systems. Utilities include a screen format generator, message file builder, data file utility, sort utility, source edit utility, terminal definition file utility, and EBCDIC-to-ASCII data file conversion utility.

ENVIRONMENTAL REQUIREMENTS

LPI compilers occupy between 1M and 2M bytes of hard disk space, depending on the language, and require 2M bytes of memory. Since processors and UNIX implementations are different, LPI provides compilers designed specifically for each customer's configuration.

TABLE 6. LPI-PASCAL EXTENSIONS

- The *extern* directive specifies that a procedure or function is defined externally.
- Optional otherwise clause for case statements.
- Use of the dollar sign ($) anywhere in identifiers.
- Binary operators & (bitwise and) and ! (bitwise or), allowing integers to be used as bit-flags.
- Underscore character (_) in any position but the first position in identifiers.
- *%include* compiler directive allows insertion of source code from a specified file.
- External files do not have to be declared as program parameters.
- RESET and REWRITE allow a second optional parameter for filename specification.
- Supports the close file handling procedure.
- $E ± compiler directive enables/disables generation of external references.
- $A ± compiler directive enables/disables subscript checking.
- $R ± compiler directive enables/disables range checking.
- $L ± compiler directive enables/disables generation of program listing.
- Defines standard text files named input and output as the terminal.
- Ignores a variant list specified in a new or dispose procedure.
- Allows the parts of a program or procedure block to appear in any order. Declaration parts can be repeated, except for forward references and duplicate declarations.

Language Processors, Inc.
Multi-Language Family

PRICING AND SUPPORT

Pricing

Pricing for LPI products running on the MC680X0, MC88000, Sun SPARC, and WE32000 processors is broken out into four categories—Class A, Class B, Class C, and File Servers.

Class A processors support up to eight I/O ports and include Convergent Technologies' S/50 and S/80, along with AT&T 3B1.

Class B processors support 9 to 32 I/O ports and include Altos 680X0; Apollo 3000/4000 single-user workstations; Arix 800/900; Convergent Technologies S/120, S/221, S/222, S/640, and S/1280; Honeywell XPS-100; Hewlett-Packard 9000/300 single-user workstations; Sun-3 single-user workstations; Unisys 5000/25, 45, and 55; AT&T 3B2 600 Series; and NEC Astra XL Family.

Class C processors support 33 to 64 I/O ports and include Arix 1200/1600; Convergent Technologies S/320, S/480, S/640, S/1280; and Unisys 5000/65, 85, and 95.

Prices for these classes and a separate classification for file servers are provided in the Software Prices section below. In addition, UNIX and Xenix products for the Intel 386 processor are also provided in the price list.

Fifteen percent of the compiler list price for LPI-Cobol, LPI-PL/1, or LPI-Basic is for the LPI run-time license, which allows distribution of a program compiled with LPI software on a computer system other than the computer system for which LPI software was purchased. There is no run-time license fee for LPI-Fortran, LPI-C, or LPI-Pascal under UNIX or Xenix.

Media

Customers specify the release media desired, according to the processor. Available media includes cartridge tape, mag tape, 5¼-inch diskettes, and 3½-inch diskettes.

Documentation

Documentation is included in compiler purchases and can be purchased separately. Each documentation set consists of a *Language Reference Manual, Users Guide, Quick Reference Guide, Release Notes,* and periodic *Technical Bulletins.* The language reference manual describes all of the features of the language as implemented by LPI. The user's guide describes the operation of the compiler on a particular system and includes the command sequences needed to invoke the compiler and compilation options. Release notes outline the product's implementation and installation status.

Support

All LPI products are provided with 30 days of introductory *CareWare* service. After this period, three levels of increasing CareWare service products are available. CareWare One provides responses to written requests, a 25 percent discount off the update price of a compiler, and a subscription to the CareWare Technical Bulletin containing technical update information. CareWare Two incorporates all CareWare one provisions, includes telephone technical support and a 50 percent discount off the compiler update price. CareWare Three is offered as a supplement to CareWare Two and provides up to 40 hours of in-depth technical consultation services during a three-month period (used for application conversions or UNIX consultation).

SOFTWARE PRICES

	Price ($)	Update ($)
LPI-Basic		
Class A	695	345
Class B	1,695	845
Class C	2,995	1,495
File Server	5,995	2,997
80386 Processor UNIX	695	345
80386 Processor Xenix	695	345
LPI-C		
Class A	695	345
Class B	1,695	845
Class C	2,995	1,495
File Server	5,995	2,997
80386 Processor UNIX	695	345
80386 Processor Xenix	695	345
LPI-Cobol		
Class A	795	395
Class B	1,995	995
Class C	3,595	1,795
File Server	8,995	4,497
80386 Processor UNIX	1,495	745
80386 Processor Xenix	1,495	745
LPI-Fortran		
Class A	795	395
Class B	1,995	995
Class C	3,595	1,795
File Server	6,795	3,397
80386 Processor UNIX	995	495
80386 Processor Xenix	995	495

Language Processors, Inc. Multi-Language Family

	Price ($)	Update ($)
LPI-Pascal		
Class A	695	345
Class B	1,695	845
Class C	2,995	1,495
80386 Processor UNIX	995	495
80386 Processor Xenix	995	495
LPI-PL/I		
Class A	995	495
Class B	2,495	1,245
Class C	4,495	2,245
File Server	8,995	4,497
80386 Processor UNIX	1,995	995
80386 Processor Xenix	1,995	995

LPI-RPG II Development System

Includes all components necessary to develop and run the application on the system for which the compiler was purchased.

	Price ($)	Update ($)
Class A	1,995	995
Class B	3,995	1,995
Class C	5,995	2,995
80386 Processor UNIX	2,995	1,495
80386 Processor Xenix	2,995	1,495

LPI-RPG II Execution Fee

This provides rights to incorporate the LPI utilities and run-time library modules necessary to run applications on a system other than the one for which the compiler was purchased.

	Price ($)	Update ($)
Class A	995	495
Class B	1,295	645
Class C	2,295	1,145
80386 Processor UNIX	995	495
80386 Processor Xenix	995	495
CodeWatch (LPI-Debug)		
Class A	495	245
Class B	1,195	595
Class C	2,495	1,245
File Server	5,995	2,997
80386 Processor UNIX	795	395
80386 Processor Xenix	795	395
CoEdit		
80386 Processor UNIX	349	175
80386 Process Xenix	349	175

MAINTENANCE FEES

	Price ($)
Introductory Careware	
Thirty days free	0
CareWare One	
Includes CareWare Technical Bulletin, 25% off software update price, and responses to written requests.	150
Careware Two	
Includes CareWare Technical Bulletin, 50% off software update price, and telephone support.	500
CareWare Two for LPI-RPG II	
Includes CareWare Technical Bulletin, 50% off software update price, and telephone support for LPI-RPG II.	1,500
CareWare Three	
Includes up to 40 hours of consultation services within three months. Must be purchased with CareWare Two. □	4,000