ABSTRACT

This paper describes the major features of the SAS/C® compiler and library and discusses the development history of the product. The enhancements in the current production release are outlined. Some possible enhancements for future releases are described.

INTRODUCTION

The SAS/C product has had four production releases in its five years of development. Each release included many new features and enhancements of existing features. Not only is the compiler used exclusively for Version 6 of the SAS® System on IBM® mainframe hosts, it is the leading C compiler in this market. Although the compiler and library are heavily oriented to use in large software systems, it is an efficient tool for any software project that is written in the C language.

The primary elements of the SAS/C product are the compiler and run-time library. However, the product includes a number of utility programs, as well as several configurations of the run-time library for specialized environments. Because the documentation is as important as the software itself, the SAS/C product is accompanied by a complete and detailed set of manuals and technical reports.

This description begins with a history of SAS/C development that highlights the features of the first three releases. Following this is an overview of the product as it is today. The last part is a discussion of what directions the product may take in the future. Throughout, features exclusive to the SAS/C product are highlighted, as are those features that were implemented in response to requests from our users.

HISTORY OF THE SAS/C COMPILER AND LIBRARY

In mid-1984, SAS Institute Inc. decided to develop its own C compiler. This project was assigned to the Institute’s Language Systems Department. The goal of this group was to make Lattice® Inc.’s compiler work on the IBM 370 architecture under the MVS/370, MVS/ESA®, and VM/CMS operating systems. A little over a year later, the progenitor of the SAS/C product was put into production.

The following sections describe each production release of the SAS/C product. Specific features are mentioned in the section covering the release in which the feature appeared. Of course, all of the features have continued to be supported in newer releases.

Release 2.10C

The first production release was known as the Lattice Native Compiler as Modified by SAS Institute Inc. for IBM 370 Systems. The compiler was based on Version 2.00 of the Lattice C compiler. The run-time library was a new implementation of the standard C library, with emphasis on both IBM suitability and compatibility with UNIX® implementations.

An Overview of the SAS/C® Compiler and Library
Tim Hunter, SAS Institute Inc., Cary, NC

The compiler implemented the C language as described in the then definitive book on C, Kernighan and Ritchie’s The C Programming Language. Because the compiler was implemented for IBM 370’s, it included many features that were thought necessary for that architecture and for programmers who were used to working in that environment. The following list describes some of these features:

- The generated code is fully reentrant, even allowing modification of external variables.
- The compiler supports a number of built-in functions. These are functions for which the compiler generates instructions directly into the instruction stream, rather than generating a call to a separately linked version of the function. In this release, the following functions were implemented as built-in functions:
  
  ```
  strcmp  memcmp  abs
  strcpy  memcmp  ceil
  strncpy  memset  fabs
  memset  strict  floor
  memcpy  bzero  ldexp
  ```

- The compiler produces a source listing file, including macro expansion and cross-reference.
- C programs can call, and be called by, programs written in IBM 370 assembly language using a very simple interface. Many assembler subroutines require no modification.
- The INDEP compiler option generates code that allows programs to execute without the need for the run-time library, and that can be called from other high-level languages such as PL/I, COBOL, and FORTRAN. (Exclusive.)

The compiler also supports several extensions that were deemed important to the usability of the C language in the IBM 370 environment. Some of these extensions are described in the following list:

- The characters ‘\1’, the circumflex are commonly used in C programming, but are not always available on IBM terminals and printers. Therefore, the compiler supports digraphs (two-character sequences) as substitute representations of these characters. You can specify up to four representations of each character, two per use in the source file and two for use in the listing file. The alternate representations may be modified on-site.
- In order to allow the generation of call-by-address parameter lists, the compiler supports the @ special operator. When applied to a function argument, this operator, like the & operator, returns the address of the argument. Unlike the & operator, the @ operator accepts an argument that is not an array, in which case the @ operator forces the creation of a temporary copy of the argument and returns the address of the copy. (Exclusive.)
In order for C programs to call assembly language functions that expect a VL-format parameter list, the compiler supports the ____asm... prefix for function names. When a function whose name is prefixed by ____asm... is called, the compiler creates a VL-format parameter list for the function. (Exclusive.)

The Run-Time Library

As mentioned earlier, the design of the run-time library had two goals, suitability for IBM 370 operating systems and compatibility with UNIX implementations. Most of the library functions are standard functions; that is, they are functions that most C implementations support. These functions can be grouped as follows:

- memory allocation functions
- character-type functions
- program control functions
- string functions
- mathematical functions
- data and time functions
- variable argument list functions.

Because the UNIX I/O model (widely used in the existing implementations of the C library) is so different from the IBM 370 model, the library I/O subsystem was implemented only after much thought and hard work. The final result was three basically separate sets of I/O functions: standard, augmented, and UNIX-style. Standard I/O functions are those commonly used, such as fopen, printf, and scanf. Augmented I/O functions (exclusive) are closer to IBM 370 I/O models and include afopen, afread, and afwrite. The UNIX-style I/O functions are close approximations of the UNIX operating system-level I/O routines open, read, write, and lseek.

The library also contains some functions designed specifically for use in the IBM 370 environment, such as the dynamic loading functions load and unload. These functions allow C programs to be implemented as several distinct load modules, each of which can be loaded into memory as required and unloaded when it is no longer needed. Other functions are present that follow common IBM 370 idioms (for example, the strchr and strcmp functions for string translation).

In addition, a version of the library that allows execution in other operating systems is provided. This version, called the Generalized Operating System environment, does not use operating system services directly. Instead, the library invokes one of several exits. Each exit is coded for use in the target operating system, and, as such, may take whatever action is appropriate to fill or deny the library's request. (Exclusive.)

Finally, most of the run-time library was packaged as a set of transient load modules. This packaging allowed individual C programs to take up less space on disk, simplified system maintenance, and allowed the library to be installed in a shared area, such as the LPA or a DCSS.

Utilities

Two utility programs, CLINK and OM370, were included in the Release 2.1OC package. CLINK is an object code preprocessor that is used when a reentrant program consists of two or more object decks produced by the compiler. CLINK combines the data needed to initialize external objects and ensures that all references to an external object refer to the same object. OM370 is an object code disassembler that, given an object deck produced by the compiler, produces a listing similar to that produced by an assembler, optionally interspersing C language source lines at the appropriate points in the listing.

Release 3.00F

Shortly after Release 2.1OC went into production, the Language Systems Department began converting Lattice's Version 3.00 compiler. As shown below, the new release contained several important enhancements. Release 3.00F, the first release to use the SAS/C name, went into production 13 months later, December 1986.

The Compiler

Both Lattice, Inc., and the Institute had been members of the ANSI X3J11 committee (the committee responsible for producing a standard for the C language and library) for some time. Therefore, Release 3.00F contained several new language elements. Some of these elements are

- the void data type
- structure assignment, structure arguments, and functions returning structures
- the enum data type
- function prototypes
- the __LINE__ and __FILE__ pre-defined macro names.

Also, the compiler implemented the register storage class. Up to six general purpose registers may be assigned to auto integer and pointer variables, and up to two floating-point registers may be assigned to floating-point variables.

The Run-Time Library

The run-time library gained new functions in several areas. These functions are

- the string functions memchr, memlwr, strlwr, strupr, strlwr, and 14 others.
- a set of system interface functions including cuserid and functions that allow access to operating system information such as the name and release numbers.
- two new utility functions, bsearch and qsort.

This release also defined a subset of the library informally called curtail. This group of functions, including the math and string functions, are independent of operating system services. As such, they were intended for use in programs that execute without the full library.

Utilities

The GENCSEG utility was added in Release 3.00F, also. The utility program allows dynamically-loadable C load modules to be saved in a sharable VM segment. This is the preferred method of installation for large applications users. Such applications execute faster and use less memory in the user's virtual machine.
Release 3.01F

Release 3.01F, the third production release, became available in March 1988. SAS/C user feedback grew proportionally with the number of SAS/C users; therefore, an important tradition was established in this release—the inclusion of compiler and library enhancements based on such feedback. Following the Institute’s lead of responding quickly to input from users of the SAS System, the SAS/C developers began implementing those items the users requested. For the remainder of this paper, this sort of enhancement will be highlighted.

The Source-Level Debugger

The most popular feedback request was for a source-level debugger, so Release 3.01F included one. Some of the features of the debugger are:

- breakpoints at every source line and at every function call, entry, and return
- single-step mode
- ability to display and modify the values of both scalar and aggregate objects
- ability to display a traceback (for example, a list of the functions in the calling sequence and the line number where each function was called)
- ability to resume execution after a program check
- ability to issue an operating system command from the debugger
- ability to operate in tandem with an operating system debugger such as TSO TEST or VM PER (exclusive)
- ability to execute a CLIST or EXEC containing other debugger commands or operating system commands (exclusive)
- ability to list the source code by line number or function name
- I/O exits that allow routing of debugger I/O to and from alternate sources and destinations, or allow the I/O to proceed via a non-standard path. (Exclusive. This facility was added by user request.)

The Compiler

Several new ANSI features were added to the release, as well as a number of language extensions requested by the users.

- non-reentrant code generation (user request).
- the void = data type and the const and volatile qualifiers.
- in-line machine code. This feature allows the C program to specify an exact sequence of IBM 370 machine instructions to be generated directly in the generated code. Almost any instruction can be specified, including exotic instructions that would not normally be generated by C language statements. (Exclusive).
- enhanced strcpy and strlen built-in functions (user request).
- Three new built-in functions, gets, printf, and strftime.

The Run-Time Library

The run-time library gained five major subsections, as well as numerous other new functions.

- A set of signal functions that go far beyond the level of support required by the ANSI draft standard. Synchronous signals, such as overflow, and asynchronous signals, such as terminal interrupts, can be trapped and handled as required by the C program. Up to sixteen user-defined signals can be added. (Exclusive).
- An important extension to signal-handling is support for IUCV signals in programs running in VM (exclusive).
- Subcommand processing functions that allow a C program to get input from subcommands contained in a CLIST or EXEC (exclusive).
- Support for Rexx in CMS, including the ability to create a Rexx function package containing C functions that can be called as a Rexx function or subroutine (exclusive); the ability to get, set, or drop Rexx variable values; and the ability to put lines on the stack in either LIFO or FIFO order (user request).
- A set of coprocessing functions that allow a C function to execute as a set of cooperative processes (exclusive).
- CMS low-level I/O functions that interface directly with the CMS file system macros, and a set of XEDIT I/O functions that allow a program to read and write directly to a file in XEDIT.

The ANSI fseek and fgetpos functions were added to the library, allowing random access to most file types. These functions were added by the ANSI committee in order to provide a more general method of file positioning than had been allowed by the fseek function. The library I/O functions were enhanced to allow I/O to VSAM files, including ESDS and KSDS file types. In CMS, the I/O functions were enhanced to allow input to come from a file in XEDIT.

Documentation

Release 3.01F was accompanied by a set of four manuals. These manuals remain the standard documentation for the SAS/C products. They are:

SAS/C Compiler and Library User’s Guide
SAS/C Library Reference, Volume 1
SAS/C Library Reference, Volume 2
SAS/C Source Level Debugger User’s Guide

RELEASE 4.00C

Release 4.00C of the SAS/C product went into production in February of this year. This is the first mainframe release to be entirely developed at the Institute since the acquisition of Lattice, Inc. The Language Systems Department became the C Compiler Development Division. The division was expanded to include developers for both the PC compiler and the mainframe compiler, and includes a complete testing department.

The major enhancements to the product are complete C language conformance to the latest draft ANSI standard, extensive support...
The Compiler

The major change to the compiler was complete conformance to the current draft ANSI standard. In addition, a global optimizer was added to the compiler as well as several language enhancements.

The Global Optimizer

While previous releases of the compiler have always emitted highly optimized machine code, the compiler was largely limited to optimizing small sequences of statements. However, with the addition of the global optimizer, the flow of control and data through an entire function can be analyzed, resulting in considerable savings in code space and execution time. Some of the optimizations performed are listed below:

- Heavily used variables are automatically allocated to machine registers. The global optimizer will allocate up to six auto integer and pointer variables to general registers, and up to two floating point variables to floating point registers. Although the C language allows variables to be given the register storage class, this declaration does not allow the variable to be assigned to a register at one point in a function and not at another point. This is less than optimal if the variable is heavily used at some places and not in others. The global optimizer will change a register's assigned variable in mid-expression, if necessary, in order to keep the most heavily used variables in machine registers throughout the function.

- Assignments that are not referenced later, called dead stores, are eliminated. The global optimizer keeps a record of references to all of the variables in a function. If a value is assigned to a variable, but that variable is never referred to later, then the code to make the assignment is not generated.

- Invariant calculations are moved out of loops. If an expression (for example an array reference with a constant index) is found in a loop and its value does not change within the loop, it is moved out of the loop.

- Common subexpressions are merged.

- Constants are propagated and any resulting constant expressions are folded. If a variable is assigned a constant value and not modified throughout the function, the global optimizer replaces references to the variable with the constant. If the variable is the wrong type, the global optimizer creates a constant of the correct type. For example, if a constant int variable is only used in float expressions, then the constant will be represented as a float constant.

- Unused code is eliminated. If the global optimizer's analysis of control flow through the function shows a section of code that cannot be executed, it will remove the code.

- Very busy expressions are hoisted. Expressions that are computed along all paths from a given point in the function are called very busy expressions. If the global optimizer detects a very busy expression, it moves (hoists) the computation to a single, common location.

Language Extensions

Several useful extensions to the language were for Release 4.00C. All of these extensions are exclusive to the SAS/C product. They include:

- the ability to declare a union with no tag and no identifier, known as an anonymous union. Anonymous unions can greatly simplify references to objects with complex definitions, as well as change a simple member to a union.

- the ability to declare an array with zero members in a structure. The array occupies no storage, but the name of the array can be used in expressions, and the member following the array is correctly aligned for the array type.

- the __asm__ keyword. This keyword may be used in the declaration of a function or function pointer. It indicates that the compiler should assume that the function is written in assembly language and should generate an appropriate function call and parameter list.

- the ability to declare char, short, or long bit fields. Additionally, the default type (either char, short, or long) of an int bit field may be specified via an option. The memcpy, memset, and memcmp built-in functions have been enhanced so they generate the smallest code sequence possible based on the type of the length parameter. If the type is char, for example, then only the code necessary to operate on 255 or fewer characters will be generated. (This was added in response to user request.)

Interlanguage Communication

Release 4.00C contains a new feature called Interlanguage Communication, or ILC. This feature is a combination of compiler enhancements, run-time library enhancements, and a new linking utility called ILCLINK. Together, these items allow C programs to call, and be called by, programs written in other high-level languages. The languages supported are FORTRAN, PL/I, COBOL, and Pascal. It is also possible for a site to add support for additional languages.

ILC support allows a programmer to freely mix routines written in any of these languages in a program. Some of the benefits of doing this are:

- using existing subroutines written in another language

- using the language most natural for a specific part of the application

- easy conversion of a program from one language to another.

The SAS/C product's Interlanguage Communication feature surpasses the mixed language support found in some other high-level languages, because it allows each language's environment to be responsible for the behavior of that part of the program. For example, PL/I ON conditions are handled by PL/I, and C signals are handled by C. Other high-level languages may require the programmer to disable all exception handling in both languages. If the other language has its own debugger, both it and the SAS/C source-level debugger can be used simultaneously to debug the program.
The Run-Time Library

The run-time library has been enhanced in many areas, but the two most important are the additions of an all-resident library and the Systems Programming Environment (SPE).

The All-Resident Library

Many users requested that the Institute provide a version of the run-time library that could be linked into a C program. In response, an all-resident library has been provided that supplies this function. Linking the C program with the all-resident library results in a stand-alone load module that contains all of the support routines necessary for the program to execute. Such load modules are easier to transport to another location because the transient library does not have to reside in the new location.

Linking the program with the all-resident library is largely automatic. However, the programmer may choose to tailor the process in order to include necessary parts of the library and exclude those deemed unnecessary. The tailoring process is driven by a C include file named resident.h. When a C source program containing resident.h is compiled and the resulting object code is linked with the program, the selected routines from the all-resident library are included in the resulting load module.

The Systems Programming Environment

The C language started out as both a general programming language and as the systems programming language for UNIX operating systems. While it has been successful as a general-purpose application language on the mainframe, so far little attempt has been made to use it as a replacement for assembly language in systems programming applications. The Systems Programming Environment is intended to support such applications.

SPE is essentially a specialized version of the run-time library. This version may be grouped into five sets of support routines:

- general routines for program start-up, execution, and termination. This includes program entry, command-line processing, stack and heap memory allocation, and program exit.
- synchronous and asynchronous interrupt handling routines. A C function may be entered in response to an interrupt.
- special versions of functions found in the full C run-time library. These functions, such as malloc and free, have been written especially for the SPE library.
- interfaces to operating system services such as SVCs.
- a subset of the functions available in the full C library. These functions are those that depend upon few or no operating system services. They include string functions, math functions, and low-level I/O functions.

Most of the routines are provided in source code and may be modified as necessary for a specific environment or application.

Separate from, but important to, SPE is the DSECT2C utility program. Most systems programming applications require access to system data whose definition is only available via an assembly language DSECT. In the C language, such data should be mapped using a structure. Transforming these DSECTs into C structures is a tedious process. If many DSECTs need to be converted, this transformation is such an overwhelming process that it is easier to write the application in assembly language. Therefore, Release 4.00C of the SAS/C product includes the DSECT2C utility. DSECT2C automatically transforms any assembly language DSECT into the equivalent C structure definition. As part of the transformation,

- assembly language data types are converted into C data types
- objects with overlapping definitions are converted into unions
- assembly language EQU instructions are converted into C language macro names
- C language macro names are produced for each assembly language symbol
- a cross-reference is produced. The cross-reference section shows the assembly language symbol, its offset, length and data type, the associated C data type, and the C identifier.

Other Library Enhancements

In addition to an all-resident library and SPE, two other enhancements to the run-time library deserve special mention:

- New functions have been added. All of the following functions conform to the draft ANSI standard:

```c
assert atexit
atof atof
assert strlen
strcat strlen
assert strcat
strchr strlen
assert strtok
strcat vsprintf
assert vfprintf
strcat vsprintf
assert vsprintf
strcat vsprintf
```

- All C I/O functions support the reading and writing of VSAM linear data sets in MVS/ESA using the data-in-virtual (DIV) facility.

The Source-Level Debugger

For Release 4.00C, the source-level debugger includes the following enhancements:

- the MCnITOR command, used to interrupt the program when a data object, such as a simple variable or structure member, changes value. (This command was implemented in response to user requests.)
- the STORAGE command, used to display memory use summaries and to diagnose memory overlays caused by program bugs.
- the WHATIS command, which displays the type of a variable.
- the size of the debugger symbol table file has been reduced by as much as 30-50%.
- a reference to an object can specify that the type of the object is defined in another function in the calling sequence.

The Full-Screen Support Library

One of the most frequently requested items has been support for full-screen programming. Our response is the Full-Screen Support Library (FSSL), a collection of functions that provide a high-level interface for developing full-screen applications on mainframe systems. Its capabilities include

- an interface to IBM data stream programming for 3270 devices
• generic or abstract screen manipulation
• complete portability between MVS and CMS
• 3270 device independence.

FSSL provides functions to
• initialize and terminate a full-screen session
• define screen viewing areas, known as panels
• define individual fields within the panels
• display and read data in the panels.

Documentation
Release 4.00C is accompanied by four new manuals.

SAS Technical Report C-106, Changes and Enhancements to the SAS/C Compiler, Release 4.00
SAS/C Compiler Interlanguage Communication Feature User’s Guide
SAS/C Full-Screen Support Library User’s Guide

FUTURE DIRECTIONS
Given that the SAS/C product is now in its fourth production release in its five years of life, it is not hard to predict that more new features will be added in future releases. The task of the C Compiler Development Division is to select those additions which will prove the most useful to you. As in the past, your input will help us make these decisions. The Institute has a number of mechanisms in place for this sort of input, the most popular of which is the SASware Ballot.

Some directions are clearly indicated, others need more input. A number of possibilities are discussed in the following paragraphs. Remember that, unless specifically noted, the mention of a possible feature does not represent a promise on our part to implement the feature. These items are noted solely as points for discussion.

The Compiler
Now that the language accepted by the compiler is defined by the ANSI draft standard, it might seem that this part of the product could become stable. However, the draft standard specifies mechanisms by which implementation-dependent language extensions may be added. For example, the _asm keyword follows those specifications for the addition of new language keywords. Clearly there is room for new language extensions. One continually popular request is for a packed-decimal data type. Other popular SASware Ballot items are vector code generation and assembler source code generation.

The second most popular suggestion in this year’s SASware Ballot was for a C++ compiler. The C++ language is becoming extraordinarily popular, and so we will do our best to provide one in a future release.

The Run-Time Library
In the next major release, the library will conform to the draft ANSI standard.

The Institute has received many requests for support for exploitation-mode execution under VM/XA SP. This support will be available in the next major SAS/C release.

The Source-Level Debugger
One of last year’s most popular SASware Ballot items was a suggestion that the debugger support expressions in its commands. This enhancement has been added in the next major release.

The top vote-getter in this year’s SASware Ballot was for a full-screen debugger. This will be added to SAS/C software in a future release.

Documentation
Currently, the Publications Division plans to issue a new edition of the four basic SAS/C manuals with the next release. This new edition will incorporate all existing material along with the new material from SAS Technical Report C-106. Of course, it will cover all of the new features and enhancements.

SUMMARY
Table 1 summarizes all of the features discussed in the preceding sections. Each feature is marked with an X under the release in which it was introduced and any releases in which it was enhanced.

REFERENCES

SAS/C and SASware Ballot are registered trademarks of SAS Institute Inc., Cary, NC.

IBM is a registered trademark of International Business Machines Corporation, Armonk, NY.

Lattice is a registered trademark of Lattice, Inc.

MVS/XA is a trademark of International Business Machines Corporation, Armonk, NY.

UNIX is a registered trademark of AT&T.
Table 1 SAS/C Features Listed by First Release

<table>
<thead>
<tr>
<th>Feature</th>
<th>2.10C</th>
<th>3.00F</th>
<th>3.01F</th>
<th>4.00C</th>
</tr>
</thead>
<tbody>
<tr>
<td>__asm keyword</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>LINE</strong> and <strong>FILE</strong> predefined macros</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>break and qsort functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>const and volatile type qualifiers</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>enum data type</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>fsetpos and fgetpos functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>register storage class</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user-added signals</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>void data type</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access to files in XEDIT</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>All-resident library</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Anonymous unions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ANSI language conformance</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Augmented I/O functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Built-in functions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Call-by-address parameter lists</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Communication with assembly language</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Complete C library</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Coprocessing functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Cross-reference listing</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLINK object code preprocessor</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CMS low-level I/O functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Digraphs for special characters</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>DIV support</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dynamic loading functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>ESMCT2C</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Full-screen support library</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Generalized Operating System library</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Global optimization</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>GENCS2</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>In-line machine code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independent execution</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Interlanguage communication</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>IUCV support via signal</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>MONITOR debugger command</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Non-int bit-fields</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Non-reentrant code</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Object Module Disassembler</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operating system interface functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>OS low-level I/O functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Prototypes</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Furtlib library</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Reentrant code</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Rexx support functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Signal functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Source listing</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Source-level debugger</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>String translation functions</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

646
<table>
<thead>
<tr>
<th>Feature</th>
<th>Release introduced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structure assignment, structure arguments, and structure return values</td>
<td>X</td>
</tr>
<tr>
<td>Subcommand functions</td>
<td>X</td>
</tr>
<tr>
<td>Systems Programming Environment library</td>
<td>X</td>
</tr>
<tr>
<td>STORAGE debugger command</td>
<td>X</td>
</tr>
<tr>
<td>Transient library</td>
<td>X</td>
</tr>
<tr>
<td>Upper- and lower-case string functions</td>
<td>X</td>
</tr>
<tr>
<td>UNIX-style I/O functions</td>
<td>X</td>
</tr>
<tr>
<td>VL-format parameter lists</td>
<td>X</td>
</tr>
<tr>
<td>void * data type</td>
<td>X</td>
</tr>
<tr>
<td>VSAM linear data set I/O</td>
<td>X</td>
</tr>
<tr>
<td>VSAM I/O</td>
<td>X</td>
</tr>
<tr>
<td>WHATIS debugger command</td>
<td>X</td>
</tr>
<tr>
<td>Zero-length arrays</td>
<td>X</td>
</tr>
</tbody>
</table>