Debunking the Myths About Fortran
Craig T. Dedo

May 16, 2011 .

# Debunking the Myths About Fortran.

Over the last few decades, there has been a lot of misinformation about the Fortran programming language. Very often, various people, many of whom should know better, state these myths as absolute truth, in spite of obvious evidence to the contrary. These myths and others like them are very widespread. I have heard each of these myths at least once in my personal experience.

This article will list and demolish these myths. If you are wondering if this effort is even necessary, please see the last myth in this article.

References to the ANSI and International Standards are by section number unless otherwise indicated. References to all other published works are by page number. References to private e-mail are by the sender's name. A list of references is at the end.

The information in this document is on a best effort basis. I actively solicit additions, corrections, and clarifications. Thanks from the author to all who contributed. Contributors are listed at the end.

A shorter version of this article appeared in the August 1999 issue of the *Fortran Forum*.

**Summary of the Myths** Following is a summary of the myths discussed in this article.

1. The only real Fortran is FORTRAN 77 (or, FORTRAN 66).
2. Modern Fortran compilers are prohibited from supporting features deleted from the Fortran standard.
3. The proper spelling of the language is FORTRAN, with all caps.
4. Fortran requires NAMES and SOURCE TEXT to be all UPPER CASE.
5. Fortran requires names to be no longer than 6 characters.
6. Fortran requires implicit typing of variables.
7. Fortran has default initialization for all variables. Numeric variables are initialized to 0, character variables to all blanks, and logical variables to .FALSE.
8. The largest size of integer is the same as the address size.
9. Fortran statements must start in column 7 and not extend past column 72.

10. Fortran is three-way IF statements and one-trip DO loops.

11. Fortran does not include modern structured programming constructs. You have to use lots of GO TO statements in order to get the control flow that you want.

12. Fortran encourages unstructured code and spaghetti-style control flow.

13. Fortran is not portable. If you really want portability, you should write your program in C.

14. Fortran is a dead language. No one uses it any more to write important applications.

15. Fortran is only for scientific and engineering applications.

16. You can't do character operations easily in Fortran.

17. You can't do bit operations in Fortran.

18. Only Microsoft makes a PC-based Fortran compiler of any importance.

19. Fortran can't do Windows.

20. Fortran can't call operating system routines.

21. Fortran can't perform database operations.

22. All Fortran compilers are expensive.

23. There are few programmers who know Fortran.

24. Fortran is no easier to learn than other popular programming languages such as C, C++, C#, or Java.

25. It is hard to find Fortran expertise.

26. A program written in Fortran is unmaintainable.

27. Nobody believes these 1 myths any longer.

**Myth: The only real Fortran is FORTRAN 77 (or, FORTRAN 66).**

The current International Standard, which is authorized and published by the International Standards Organization (ISO), is ISO/IEC 1539-1:2010, known as Fortran 2008. Under ISO rules, Fortran 2008 automatically cancels and replaces the previous International Standard, ISO/IEC 1539-1:2004, Fortran 2003 ([ISO 2010], p. xii).

FORTRAN 77 is no longer standard anywhere in the world. ANSI withdrew FORTRAN 77 as an American National Standard in 1995.

Right now, PL22.3 and WG5, the ANSI and ISO Fortran Standards Committees respectively, have recently completed the current revision of the International

Standard. This revision is referred to as Fortran 2008. Fortran 2008 was published by the ISO on October 6, 2010.

Fortran 90 is a complete superset of FORTRAN 77, with only four exceptions ([ISO 1991], 1.4.1).

Fortran 95 has only six exceptions to FORTRAN 77 ([ISO 1997], 1.5.2). The only important one is that Fortran 90 and 95 include a lot more intrinsic procedures (i.e., subroutines and functions that are included with the language) than FORTRAN 77 did.

Fortran 95 deletes five (5) obsolescent features ([ISO 1997], 1.5.1, B.1). Almost all Fortran 95 compilers still support all of these features.

Fortran is one of the few programming languages defined by an International Standard that has a regular schedule of frequent and substantial revisions and enhancements. Consider:  Fortran 90 was published in June 1991. Fortran 95 was published in December 1997. Fortran 2003 was published in November 2004. Fortran 2008 was published in October 2010. Each of these revisions contains substantial new features over all previous versions. Fortran 90 and Fortran 2003 are considered major revisions. Fortran 95 and Fortran 2008 are considered minor revisions. However, both Fortran 95 and Fortran 2008 included a few major new features. John Reid, the Convenor of WG5, has written papers describing the new features of Fortran 2003 and Fortran 2008 ([Reid 2003] and [Reid 2008]).

Fortran 95 compilers are quite real. Although a lot of older code is written in FORTRAN 77, this code can still be compiled using Fortran 95 compilers, with only a few minor modifications. Most modern Fortran 95 compilers have many Fortran 2003 features and most are also starting to include some Fortran 2008 features. One or two Fortran compilers have complete implementation of all Fortran 2003 features.

**Myth: Modern Fortran compilers are prohibited from supporting features deleted from the Fortran standard.**

The concept of language evolution was introduced in the FORTRAN 77 standard, which was the second ANSI and ISO Fortran standard. FORTRAN 77 removed or redefined several features. Two of the more prominent changes in FORTRAN 77 were the replacement of Hollerith character data with the CHARACTER data type and the requirement that a DO loop with a zero or negative loop count shall not be executed.

The Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008 standards continue the concept of language evolution ([ISO 2010], 1.6 and 1.7). Annex B contains a list and description of all features that have been deleted from any Fortran standard since Fortran 90 ([ISO 2010], B.1). It does not include those features that were deleted from the FORTRAN 66 standard when the FORTRAN 77 standard was published. Annex B also includes a list and description of the features that were declared to be obsolescent in Fortran 2008 or earlier ([ISO 2010], B.2). Obsolescent features are still part of the current Fortran standard but may be deleted from a future Fortran standard.

A feature deleted from the Fortran standard has exactly the same status as a feature that has never been included in any edition of the Fortran standard. It is an "additional form or relationship that is not permitted by the numbered syntax rules or constraints" ([ISO 2010], 1.5). The deleted features are expressly included as additional forms or relationships. The Fortran standard explicitly allows a standard-conforming processor to include additional features.

Almost all vendors of Fortran compilers, responding to customer demand, support all features that have been deleted from any version of the Fortran standard.

## Myth: The proper spelling of the language is FORTRAN, with all caps.

Originally, the word FORTRAN was an acronym, for FORmula TRANslation. The word and its origin were explicitly stated in the title of John Backus's original 1954 paper, "Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN" ([IBM 1954], 1-2).

By official ruling of the ISO, the language name was changed to a proper noun, "Fortran". This is explicitly stated in the Introduction to the Fortran 90 standard ([ISO 1991], p. xiii): "Note that the name of this language, Fortran, differs from that in FORTRAN 77 in that only the first letter is capitalized. Both FORTRAN 77 and FORTRAN 66 used only capital letters in the official name of the language, but Fortran 90 does not continue this tradition." Unfortunately, subsequent Fortran standards do not include this paragraph.

## Myth: Fortran requires NAMES and SOURCE TEXT to be all UPPER CASE.

Prior to Fortran 90, the standard only mentioned upper case. However, most of the older FORTRAN 77 compilers allowed both upper case and lower case whenever both cases were available on the host hardware and operating system (James Giles). All currently available FORTRAN 77, Fortran 90, Fortran 95, and Fortran 2003 compilers support both upper case and lower case syntax.

Fortran 90 and Fortran 95 explicitly allow, but do not require, the support of lower case in syntax elements. Both standards require that Fortran syntax is not case sensitive ([ISO 1997], 3.1.1).

Fortran 2003 and Fortran 2008 require Fortran compilers to support both upper case and lower case syntax ([ISO 2004], 3.1.1; [ISO 2010], 3.1.2). I should know; I wrote the paper that implements this requirement.

The requirement that Fortran syntax is not case sensitive is an advantage. Case sensitive languages are generally more error prone (James Giles).

**Myth: Fortran requires names to be no longer than 6 characters.**

The FORTRAN 66 and FORTRAN 77 standards limited names to 6 characters. However, many pre-Fortran 90 compilers extended the limit, some to 8 characters, others to as many as 31 characters.

A the same time that Fortran’s standard limited variables to 6 characters, C had no standard at all and K&R (first edition) was considered the authority. Yet, K&R stated that only the first 8 characters of an identifier were significant, even though identifiers of any length were allowed- This is appallingly bad language design. Identifiers that were intended to be different, but which were the same in the first eight characters, were silently treated as the same. Identifiers for external symbols were usually significant to fewer than eight characters (often six, since linkers were designed for Fortran). This was much worse than the enforced Fortran limit because it was not enforced. It was not eliminated until the C standard, which was about the same time as Fortran 90 (James Giles).

The Fortran 90 and Fortran 95 standards extended the limit to 31 characters and allowed the use of underscores in names ([ISO 1997], 3.2.1, 3.1.3). Fortran 2003 and Fortran 2008 extend this limit to 63 characters ([ISO 2004], 3.2.1; [ISO 2010], 3.2.2).

All currently published FORTRAN 77 compilers that I am aware of support 31 character names.

**Myth: Fortran requires implicit typing of variables.**

The Fortran implicit typing rule states that variables that start with any of the letters I, J, K, L, M, or N are integer and all other variables are real (a.k.a. floating point). This implicit typing rule was part of the original specification ([IBM 1954], 3).

Since FORTRAN 77, you could override the implicit typing rule by explicitly declaring the type of a data object ([ISO 1997], 5.1). You could also declare a different implicit typing rule. Starting with Fortran 90, you could require explicit declaration of all data objects by using the IMPLICIT NONE statement ([ISO 1997], 5.3).

**Myth: Fortran has default initialization for all variables. Numeric variables are initialized to 0, character variables to all blanks, and logical variables to .FALSE.**

Although some Fortran compilers may implement this or some other form of default initialization, this is not required by the Fortran standard. Certain classes of variables are initially defined. These include variables that are initialized in DATA statements or type declaration statements and variables that have components of a derived type with default initialization. All other variables are initially undefined ([ISO 2010, 16.6.3, 16.6.4]).

**Myth: The largest size of integer is the same as the address size.**

The Fortran standard does not have, nor has it ever had, any rule in it that specifies that the largest size of integer is the same as the address size of the hardware on which the program runs. In fact, the Fortran standard has no concept of address size at all.

The Fortran standard only requires that a Fortran compiler support one or more representation methods that define sets of values for integers ([ISO 2010], 4.4.2.2). There is no requirement that a standard-conforming compiler support any maximum size of integer. The Fortran 2008 standard specifies that a standard-conforming compiler support at least one integer kind of at least 18 digits (usually 64 bits) ([ISO 2010], 4.4.2.2).

This is not an academic issue. There are real compilers, of commercial significance, on the market today that support integers larger than the address size. The compiler may need to use some special techniques in order to implement integers larger than the address size. However, such integers are declared using the standard syntax, such declarations have the standard semantics, therefore such declarations are fully standard-conforming.

**Myth: Fortran statements must start in column 7 and not extend past column 72.**

The first part has never been true. You could always indent beyond column 7 if you chose (James Giles).

The maximum limit of column 72 is true only for the older fixed source form. Fixed source form is now on the Obsolescent Features List ([ISO 2010], 3.3.3, B.2.7).

Fortran 90 and Fortran 95 support free source form. In free source form, each source line may be up to 132 characters in length and there are no restrictions on where a statement or part of a statement may appear within a line ([ISO 2010], 3.3.2).

There are many tools available for automatically converting the older fixed source form into newer free source form. Some of these tools are free.

**Myth: Fortran is three-way IF statements and one-trip DO loops.**

The three-way IF statement, formally known as the Arithmetic IF, was developed as part of the original implementation of the Fortran programming language.

The Fortran 90 standard marked the Arithmetic IF as an obsolescent feature ([ISO 1991], 8.2.5).

Fortran 2008 still keeps it on the Obsolescent Features List ([ISO 2010], 8.2.4, B.2.1).

The one-trip DO loop was developed in 1954 as part of the original specification ([IBM 1954], 8).

In this design, if the formula for the iteration count of the loop produced a value of less than one, the loop was executed once anyway.

Keep in mind that Fortran was one of the earliest commercially successful high level programming languages. In those early days, language theory was not very well developed. Almost all of the major advances in the design of programming languages were still to be discovered (Mark Brown).

The first Fortran standard, FORTRAN 66, stated explicitly that the initial value of the control variable must be less than or equal to the final value ([ANSI 1966], 7.1.2.8.1). Thus, the iteration count had to be at least one. The behavior of a program with an iteration count of less than one was processor dependent. A large majority of Fortran compilers used the one-trip rule.

The FORTRAN 77 standard revised the DO loop iteration rules to specify that if the iteration count formula produced a value less than one, the loop is not executed ([ISO 2010], 8.1.6.6.1). The rule is unchanged in Fortran 90, Fortran 95, Fortran 2003, and Fortran 2008.

**Myth: Fortran does not include modern structured programming constructs. You have to use lots of GO TO statements in order to get the control flow that you want.**

Fortran includes an almost complete set of control constructs. All of them are properly defined block structured constructs. Each block can hold zero, one or more statements without requiring the programmer to do anything special, such as enclosing the statements in begin-end or {-} pairs ([ISO 2010], 8.1.1).

**Block IF.** This is the IF - ELSE IF - ELSE - END IF construct ([ISO 2010], 8.1.7). There can be any number of ELSE IF blocks and zero or one ELSE block. This construct was introduced in FORTRAN 77.

**Numeric DO.** This is the DO construct where the loop is controlled by a numerically computed iteration count ([ISO 2010], 8.1.6). The numeric DO construct has been in Fortran from the beginning ([IBM 1954], 7-9).

**DO WHILE.** This is the version of the DO construct where the execution of the loop is controlled by a relational or logical expression that evaluates to either TRUE or FALSE ([ISO 2010], 8.1.6). The DO WHILE statement was introduced in Fortran 90.

**DO "Forever".** This is the version of the DO statement without any loop control at all.

Normally, one or more of the statements in the loop body should test and decide whether to exit the loop ([ISO 2010], 8.1.6). The DO "Forever" construct was introduced in Fortran 90.

The **CYCLE** and **EXIT** statements provide additional means of loop control ([ISO 2010], 8.1.6.6.3, 8.1.6.6.4, 8.1.10). The Fortran 2008 standard extended the semantics of the EXIT statement to exit any named block construct, not just DO loops (8.1.10).

**CASE.** This is the SELECT CASE - CASE - END SELECT construct ([ISO 2010], 8.1.8). Unlike implementations of the CASE construct in some other languages, the Fortran CASE construct can use either an INTEGER or CHARACTER control variable. The CHARACTER CASE comparisons can compare on strings of any length. You can include ranges or lists of comparison values in one CASE block.

The only commonly used control construct which is not part of the standard language syntax is the **REPEAT - UNTIL** construct. You can easily simulate this control construct by using a DO "Forever" or DO WHILE construct with an IF test at the end of the loop.

Fortran 2003 and 2008 introduced several more control constructs. Fortran 2003 introduced the ASSOCIATE construct ([ISO 2010], 8.1.3) and SELECT TYPE construct ([ISO 2010], 8.1.9). Fortran 2008 introduced the BLOCK construct ([ISO 2010], 8.1.4) and the CRITICAL construct ([ISO 2010], 8.1.5).

**Myth: Fortran encourages unstructured code and spaghetti-style control flow.**

The design of FORTRAN 66 and its predecessors did not have any structured features except for the numeric DO construct. However, this did not mean that these early versions of Fortran practically required unstructured coding. Even in those days a programmer could emulate structured programming constructs with the careful, disciplined use of GO TO statements ([McConnell 1993], 391-394).

In contrast, modern Fortran encourages highly structured, modular program organization.

- There is a complete set of control constructs, as detailed in the answer to the previous myth.

- Three (3) of the more abominable unstructured language features have been deleted from Fortran 95 ([ISO 1997], B.1). These are: (1) Branching to an END IF statement from outside of the IF Block; (2) the PAUSE statement; and (3) the ASSIGN, ASSIGNed GO TO, and ASSIGNed FORMAT statements.

- Seven (7) other unstructured features are on the Obsolescent Features List ([ISO 2010], B.2).

These are: (1) the Arithmetic IF statement; (2) the non-block DO construct; (3) Alternate Return specifiers; (4) the Computed GO TO statement; (5) Statement Functions; (6) DATA Statements Among Executable Statements; (7) ENTRY statement.

- There is a full implementation of modules ([ISO 2010], 2.2.4, 11.2), a facility for encapsulating code and data together in a single object. This encourages the development of modular program structure. Fortran 2008 added submodules, a feature that allows the programmer to separate the module interface from the module implementation ([ISO 2010, 2.2.5, 11.2.3).

- There is support for internal procedures. This allows the programmer to develop local procedures within procedures ([ISO 2010], 12.2.2.2).

- There is support for user-defined operators and user-defined assignment. This allows a programmer to define operators for common, complex operations ([ISO 2010], 3.2.4, 7.1.6, 7.2.1.4, 7.2.1.5, 12.4.3.4.2, 12.4.3.4.3).

Much of the purported "evidence" for this myth comes from the state of a lot of early code, some of it over 30 years old. The poor organization of older code has a number of causes.

(1) Prior to 1968, little was known about how to write code clearly and well, so that humans could understand it. Our current knowledge of how to write readable, maintainable code has been the result of years of research into the psychology of

computer programming and some intense debates about programming style and control flow.

(2) In the early days, computers, main memory (RAM), and disk space were all horribly expensive. Thus, early machines had extremely limited memories. One program that I ported to a PC in 1998 was originally written for a computer with 8K of RAM and an 8K drum. Because memory was so limited, programmers often resorted to various hacks in order to fit the entire program into the available memory.

(3) Many of the early programmers were professionals in other fields, such as engineering. They did not have any professional training in how to write programs.

**Myth: Fortran is not portable. If you really want portability, you should write your program in C.**

In reality, Fortran is at least as portable as C is, in the true sense of that term.

Unfortunately, the term “portable” has two different definitions, which are as unlike as apples and oranges.

1. “Portable” means that a compiler for the language exists on just about any system and with a sufficient amount of hacking, you can get the program to work in some fashion. This is not a true definition for “portable”.

2. “Portable” means that the program can work on the target system with minimal changes to the source code, have the same semantics as it did on the source system, and produce results close or identical to the program results on the source system. This is the true meaning of “portable”.

The *Computer Glossary* defines “portable” as, “Refers to software that can easily be moved from one type of machine to another. It implies a product that has a version for several hardware platforms or has built-in capabilities for switching between them. However, a program that can be easily converted from one machine type to another is also considered portable.” ([Freedman 1995], 306) The *IEEE Software Engineering Standard* ([IEEE 1990]) defines “portability” as, “the ease with which a system or component can be transferred from one hardware or software environment to another.” Fortran is intentionally designed to be “portable” in the true sense of the word. Although there are a large number of features which the standard defines to be “processor dependent” in their implementation, these features are processor dependent in almost every other language as well.

They include such things as valid file specifications, the numbers of different kinds of each data type, etc. The Fortran 2008 standard includes a list of processor-dependent features ([ISO 2010], A).

Fortran is not tied to byte-addressable machine architecture. REAL, COMPLEX, and INTEGER data types can be any number of bits. It is even possible to define them in a way that they could exist on non-binary (e.g., decimal) machines. The same is true of LOGICAL and CHARACTER data types.

Obviously, there are C compilers available for just about any platform in existence. However, there is no dearth of Fortran compilers for particular platforms. Right now, there are full-featured Fortran 90 or 95 compilers for nearly every economically important platform that exists, with the exception of IBM mainframes. Even IBM mainframes have highly optimizing FORTRAN 77 compilers that are heavily used.

**Myth: Fortran is a dead language. No one uses it any more to write important applications.**

New applications of serious importance are being written in Fortran every day. At a PL22.3 meeting around 2000, Tony Warnock, the PL22.3 Chair at the time and a researcher at Los Alamos National Lab, told everyone that the management at Los Alamos decided to write three large, new applications in Fortran. Commercial and industrial firms with real dollars at stake often choose Fortran as the development language because of its human productivity advantages.

The biggest obstacle to greater use of Fortran is that Fortran is Politically Incorrect. Many younger programmers refuse to program in Fortran because they fear loss of status and prestige if “Fortran” appears on their resumes. Many decision makers in management are either ignorant of the technical and business advantages of Fortran or are simply following the current language fads.

In such an atmosphere it is impossible to make decisions based on bona-fide technical and business criteria.

These observations are based on my personal experience. Other people I have talked to have reported the same attitudes, especially among younger programmers. Also, there are a lot of younger programmers who post articles on the comp-lang-fortran Usenet newsgroup who ask if it is even worthwhile to learn Fortran.

**Myth: Fortran is only for scientific and engineering applications.**

Fortran is designed to have high run time efficiency for computing intensive numeric applications. Consequently, it is the language of choice for many important science and engineering simulations.

Other important design characteristics are a high level of expressive power and user-friendly syntax. These design characteristics make Fortran well suited for a wide variety of applications.

Such application areas include: - Financial analysis - Shop floor control - Industrial automation - Text processing and character string operations - Database processing I have worked in many application areas where C was the development language, by management fiat. In most of these cases, it would have been far more cost effective to use Fortran instead. A good example is shop floor control and industrial automation, either actually controlling the machinery out on the shop floor or tracking products through the production process. In either of these cases, the vast majority of the data consists of various kinds of character strings. In the case of industrial automation, almost always the Programmable Logic Controllers (PLC) on the equipment are communicating with the host computers via character strings, not binary numbers. In the vast majority of automation and production tracking applications, the data are kept in databases and the applications read from and write to the databases. Almost all of the database fields are character fields.

**Myth: You can’t do character operations easily in Fortran.**

Admittedly, early versions of Fortran had little ability to process character data and what capability existed was extremely clumsy. FORTRAN 66 stuffed characters into either REAL or INTEGER arrays. This practice was known as Hollerith data. The introduction of a real CHARACTER data type in FORTRAN 77 made Hollerith data obsolete.

Today Fortran is well suited for character operations and text processing applications. Consider the following features of the Fortran character capabilities: - The CHARACTER data type is a real, first class data type ([ISO 2010], 4.4.3). This is similar to strings in Basic. Fortran character strings are not hacked-up arrays of integers.

- Character strings can be any length, up to a processor-dependent limit ([ISO 2010], 4.4.3.1). This is often 32767 or 65535. Some compilers have larger limits.

- The derived-type facility allows a programmer to create string types of larger sizes than the compiler limit ([ISO 2010], 4.5).

- A Fortran compiler can choose to support multiple character sets. The KIND type parameter is defined for the CHARACTER data type, as it is for other intrinsic data types ([ISO 2010], 4.4.3.2).

- There are no reserved characters in any Fortran character sets that serve as magic tokens (such as the backslash and null characters in C) ([ISO 2010], 4.4.3.1).

- The relational operators (==, /=, <, >, <=, >=) work just as well on character strings as they do on numeric operands ([ISO 2010], 7.1.5, 7.1.5.5). Thus, the programmer is not required to use functions like *strncmp()*. This makes character comparisons easy, straightforward, and intuitive.

- Similarly, character assignment and concatenation are performed using operators (= and //) ([ISO 2010], 7.1.5, 7.1.5.3, 7.2.1.2) instead of functions such as *strncpy()* and *strncmp()*.

- There is no need to worry about mismatches in string length (in most cases). The semantics of Fortran character comparisons require that shorter strings be blank-extended on the right to the length of longer strings ([ISO 2010], 7.1.5.5.1). For character assignment, the value on the right side of the = sign is truncated or blank padded on the right as necessary ([ISO 2010], 7.2.1.3). This feature eliminates a lot of headaches and increases human productivity.

- Substring references are easy and straightforward, using standard (begin : end) positional notation ([ISO 2010], 6.4.1).

- String constants can be enclosed either in apostrophes (') or quotes (") ([ISO 2010], 4.4.3.3). This means that strings containing either can be represented easily with no need for escape characters or the like (Clive Page).

- Fortran 2008 adds the capability of variable-length character strings using the deferred length parameter (LEN=:) ([ISO 2010], 4.4.3.2).

- There is a complete set of character string intrinsic procedures ([ISO 2004], 13.5.3). The following table lists them and gives a brief explanation of each procedure.

| Name | Explanation |
|---|---|
| ACHAR (I) | Character in position in ASCII collating sequence |
| ADJUSTL (STRING) | Adjust left |
| ADJUSTR (STRING) | Adjust right |
| CHAR (I [,KIND]) | Character in position in processor collating sequence |
| IACHAR (C) | Position of character in ASCII collating sequence |
| ICHAR (C) | Position of character in processor collating sequence |
| INDEX (STRING, SUBSTRING [,BACK]) | Starting position of a substring |
| LEN (STRING) | Length of string |
| LEN_TRIM (STRING) | Length without trailing blank characters |
| LGE (STRING_A, STRING_B) | >= using ASCII collating sequence |
| LGT (STRING_A, STRING_B) | > using ASCII collating sequence |
| LLE (STRING_A, STRING_B) | <= using ASCII collating sequence |
| LLT (STRING_A, STRING_B) | < using ASCII collating sequence |

MAX (A1, A2, [,A3, ...]) Maximum value MAXLOC (ARRAY, [,DIM, MASK, KIND, BACK] Location(s) of maximum value <#>.

Name Explanation MAXVAL (ARRAY, [,DIM, MASK]) Maximum value(s) of array MIN (A1, A2, [,A3, ...]) Minimum value MINLOC (ARRAY, [,DIM, MASK, KIND, BACK] Location(s) of minimum value MINVAL (ARRAY, [,DIM, MASK]) Minimum value(s) of array REPEAT (STRING, NCOPIES) Repeated concatenation SCAN (STRING, SET [,BACK]) Scan a string for a character in a set SELECTED_CHAR_KIND (NAME) Select a character kind type parameter given the name of the character set TRIM (STRING) Remove trailing blank characters VERIFY (STRING, SET [,BACK]) Search for a character not in a set Although it is not strictly a CHARACTER function, a programmer can use the MERGE function to convert logical expressions to one of two strings, e.g., "YES" or "NO" (Clive Page).

The ease and efficiency of the Fortran character features can be shown by an example from my own personal experience. This example is not an academic exercise. It is a real production problem in a commercial environment, with real profit and loss responsibility, and with real money at stake.

In 1998, I did some consulting work in a paper mill. At one point I had to read a dimension for a roll of paper from a database field and properly punctuate the number. The number in the database field was in the form of iiinndd, where iii was the whole number of inches, nn was the numerator of the fractional part, and dd was the denominator of the fractional part. I had to insert a hyphen between the whole number and numerator and a slash between the numerator and denominator.

Doing this simple operation in C, using the substring extraction and concatenation functions (*strncpy()* and *strncat()*), took 8 statements and over half an hour of effort to get it right. I could have reduced the number of statements to one statement by using I/O functions like *sprintf()* (Jonathon Stott). However, using I/O features is a fairly indirect way to do simple string operations.

If I could have used Fortran, it would have taken only one statement and less than a minute of effort. Here is the Fortran code. LABEL and C are CHARACTER variables of sufficient length.

LABEL = C(1:3) // "-" // C(4:5) // "/" // C(6:7) This expresses the transformation directly and the inserted characters are in the order they appear in the result. There is no need of associating arguments to format specifiers or having the inserted characters as constants in the format specification. Further, if you wanted to do something with the string other than print it, the use of an I/O construct to do the work would be really strange. In the Fortran version, the actual operation is a character valued expression, not a part of the I/O process (James Giles).

The total time to write the Fortran version? About as long as it takes to physically type the statement into the source code.

**Myth: You can’t do bit operations in Fortran.**

The early versions of Fortran did not have bit operations. In 1978, the US Defense Department produced MIL-STD 1753, which added several bit intrinsic procedures as extensions. Due to user demand, many of the Fortran compilers of that era included all of the MIL-STD 1753 extensions.

Modern Fortran does not support an intrinsic bit string data type. However, Fortran 2003 and Fortran 2008 include all of the MIL-STD 1753 bit intrinsics ([ISO 2004], 13.5.8, 13.5.9; [ISO 2010], 13.5). A programmer can use the defined operator feature to define most of these functions as operators ([ISO 2010], 3.2.4, 7.1.6, 12.4.3.4.2). Fortran 2008 adds 16 new bit intrinsic procedures ([ISO 2010], 13.5, 13.7).

The following table lists the bit intrinsic procedures and gives a brief explanation of each procedure.

Name Explanation

BGE (I, J) Bitwise greater than or equal to

BGT (I, J) Bitwise greater than

BIT_SIZE (I) Number of bits in the integer model

BLE (I, J) Bitwise less than or equal to BLT (I, J) Bitwise less than

BTEST (I, POS) Bit testing

DSHIFTL (I, J, SHIFT) Combined left shift

DSHIFTR (I, J, SHIFT) Combined right shift

IAND (I, J) Bitwise

AND IBCLR (I, POS) Clear bit

IBITS (I, POS, LEN) Bit extraction

IBSET (I, POS) Set bit

IEOR (I, J) Bitwise Exclusive

OR IOR (I, J) Bitwise Inclusive

OR ISHFT (I, SHIFT) Bitwise Shift

ISHFTC (I, SHIFT [,SIZE]) Bitwise Circular Shift

LEADZ (I) Number of leading zero bits

MASKL (I [, KIND]) Left justified mask

MASKR (I [, KIND]) Right justified mask

MERGE_BITS (I, J, MASK) Merge bits under a mask

MVBITS (FROM, FROMPOS, LEN, TO, TOPOS) SUBROUTINE - Copies bits from one integer to another

NOT (I) Bitwise complement

POPCNT (I) Number of one bits

POPPAR (I) Parity as 0 or 1

SHIFTA Right shift with fill

HIFTL Left shift.

SHIFTR Right shift

TRAILZ Number of trailing bits

**Myth: Only Microsoft makes a PC-based Fortran compiler of any importance.**

Microsoft no longer makes ***any*** kind of Fortran compiler at all. Microsoft's last Fortran compiler, MS PowerStation 4, was published in 1995. After that, Microsoft dropped out of the Fortran market completely.

Even prior to dropping out of the Fortran market, Microsoft showed only half-hearted support of Fortran for many years. For example, in MS Fortran 3.x and below, the maximum length of a CHARACTER variable was only 127 characters, a limit so low that it made useful work quite difficult. The MS compilers generally have a reputation of being heavily bug-ridden. In my observation, Microsoft's representative to PL22.3, the ANSI Fortran Standards Committee, did not support any major new features for Fortran at all.

There are several good quality Fortran compilers for Intel-based PCs running various flavors of Windows. Here is a brief summary, with apologies to those vendors I may have missed.

Absoft (http://www.absoft.com) of Rochester, MI, sells "ProFortran v11.1 for Windows", a complete workbench for PC users, which includes an editor, Windows-based IDE, debugger, linker, profiler, application framework, and 32-bit and 64-bit compiler. Users can also optionally purchase an add-on module of the IMSL math libraries and add-on modules of one of the GINO graphics libraries.

There are two separately developed compilers that are connected with the GNU project.

G95 (www.g95.org) has five different binary distribution packages available for both the Cygwin and MinGW “native Windows” environments. The current stable version is 0.92.

GFortran (gcc.gnu.org/wiki/GFortran) is the GNU Fortran Project that is integrated with the rest of the GNU Compiler Collection (GCC). Binary packages for both Cygwin and the MinGW “native Windows” environments are available on various distribution web sites. The current stable version is 4.5.2. Version 4.6.0 is the current development version.

Intel Corporation (http://www.intel.com), sells Intel Visual Fortran (IVF) 12.0, which was formerly known as Compaq Visual Fortran (CVF) and before that as Digital Visual Fortran (DVF). It runs on Windows XP/Vista/7 and produces executables for those environments. IVF includes the Microsoft Visual Studio development environment. This is the Microsoft-recommended migration path for Fortran PowerStation users.

Lahey / Fujitsu Fortran 95 7.2 is produced by the Lahey / Fujitsu alliance (http://www.lahey.com). It is available in two configurations: Express and Professional. The Express Edition includes the LF95 compiler, linker, debugger, and librarian. The Professional Edition includes the Visual Studio 2008 IDE shell, Fujitsu Visual Analyzer, Fujitsu Scientific Subroutine Library, Fujitsu WinFDB Windows debugger, BLAS and LAPACK libraries, and the Winteracter Starter Kit. It runs on Windows XP/Vista/7 and produces executables for Windows 95/98/ME/NT/2000/XP/Vista/7.

Numerical Algorithms Group (NAG) sells the NAG Fortran Builder for Windows, Release 5.2. It includes the NAG Fortran Compiler 5.2 plus a GUI development environment and GUI debugger.

The Portland Group (www.pgroup.com) sells the PGI Visual Fortran compiler 3 which completely integrates PGI Parallel Fortran with Microsoft Windows using Visual Studio 2008.

Silverfrost Software (http://www.silverfrost.com) produces a Fortran 95 compiler, FTN95 6.00, for PCs running Windows XP/Vista/7. It comes with the Plato IDE and includes ClearWin+, a Windows development kit.

## Myth: Fortran can’t do Windows.

All of the PC-based Fortran compilers include at least one method of accessing the Microsoft Windows API. Some compilers offer more than one method.

One of the difficulties of programming using the Windows API is that it is written in C# and assumes that the programmer will be using C# or C++ to develop the application. Therefore, all of the public interfaces of the Windows API functions use C calling conventions.

Following is a list of the methods of developing Windows applications that each compiler supports.

Absoft ProFortran supports direct calls to the Win32 and Win64 API. It also includes MRWE, a simplified Windows development environment that uses Fortran procedure calling interfaces.

Intel Visual Fortran (IVF) supports direct calls to the Win32 and Win64 API. It also includes the QuickWin development environment that previously was part of Microsoft PowerStation. IVF allows the direct linking of object modules created by IVF, Microsoft Visual Basic, and Microsoft Visual C++.

Lahey / Fujitsu LF95 supports direct calls to the Win32 API. A developer can create either a Very Simple Windows program using the -VSW compiler option or else a full-featured Windows program using the -WIN option. LF95 allows the creation of DLLs that can be called from other Wintel languages. LF95 supports DLL interfaces to Microsoft Visual Basic, Visual C++, Borland C++, and Borland Delphi. LF95 supports static linking with Fujitsu C, Microsoft Visual C++, and Borland C++ object files. The Professional configuration include a subset of the Winteracter Windows development kit.

Silverfrost FTN95 supports direct calls to the Win32 and Win64 API. It also includes ClearWin+, a comprehensive Windows development kit that uses Fortran procedure calling interfaces.

There also are several third party Windows development kits that use Fortran procedure calling interfaces.

Bradley Associates, Ltd., a.k.a. GINO Graphics (http://www.gino-graphics.com) supplies GINO-F and GINOMENU. GINO-F is a professional Fortran toolkit containing over 350 low-level and high-level routines for 2D and 3D graphics. GINOMENU is a Fortran toolkit for GUI development.

Both products are available for a variety of OSes.

Polyhedron (http://www.polyhedron.com) supplies Winteracter, a Win32 Fortran 90 user interface and graphics development tool.

**Myth: Fortran can’t call operating system routines.**

In almost all real-world development projects, the developers have a substantial need to directly access various resources provided by the operating system (OS). Consequently, the providers of operating systems include one or more operating system API libraries that they intend the developers to use for making operating system calls.

Although one might expect that most Fortran applications would not need to access OS resources, the opposite is really the case. Most real world Fortran programs make extensive use of the OS resources that are available. And most vendors of Fortran compilers, responding to consumer demand, make some kind of OS interface available in their Fortran compilers.

Following is a short summary of the capabilities available under various OSes.

HP's OpenVMS provides application developers with extensive capabilities. These include sorting, parallel processing, multiple threads, inter-process communication, user-defined condition messages, date and time routines, and many other resources.

On Windows systems, the full capabilities of the Windows API libraries are available, just as they are for users of other languages. There are some interface difficulties.

Most Fortran compilers for the various Unixes offer a way to access the OS API libraries.

Harris VOS allows extensive support of calling OS-related features from Fortran. These include the support of real-time clocks, interval timers, exotic hardware, synchronous and asynchronous threads, starting and stopping programs, setting program and thread priorities, sending and receiving software and hardware interrupts, and asynchronous I/O (Gary Scott).

On IBM mainframes, VS Fortran allows full access to graphics functions (e.g., mouse, data keys, vector graphics, images, segments, object/position correlation) and OS calls (Gary Scott).

The ease with which Fortran programmers can access these OS API library routines differs with each OS. Some operating systems, such as HP's OpenVMS, use a (relatively) language-independent procedure calling standard. Other operating systems implicitly or explicitly expect that the application developer do things in the way of the operating system's source language. In many cases, the OS API is written in C and uses a C language public interface for the library routines. Since C language calling conventions are quite different from the way that most Fortran compilers usually make procedure calls, there must be some way of reconciling the differences.

Reconciling these differences in calling conventions almost always involves the creation of language syntax extensions and the provision of non-standard procedures for accessing the OS resources. Of course, this makes programs that use these features non-standard and non-portable.

However, by their very nature, OS API calls are non-standard and non-portable.

Following is a short summary of some difficulties of using OS API procedures in Fortran programs.

**Argument Passing Mechanism.** One of the areas of difference in calling interface is how a particular procedure argument is passed. Very often, OS API routines specify that certain arguments should be passed by value, by reference, or (rarely) by descriptor. In many cases, the argument passing mechanism which the Fortran compiler uses for the particular kind of argument in the call is not what the OS API routine expects. Many Fortran compilers offer extensions to specify the argument passing mechanism. One of the most common methods is to specify pseudo-functions: %VAL() for pass-by-value, %REF() for pass-by-reference, and %DESC() for pass-by-descriptor.

**Unsigned Integers.** Another problem is that many OS API procedure arguments are unsigned integers. Standard Fortran does not have unsigned integers and few Fortran compilers support them. Fortunately, in the vast majority of cases, the corresponding signed integer of the same size can be substituted without problems.

**Data Structures.** Many OS API procedure arguments require complex data structures with precise ordering and alignment requirements. Very often, these requirements differ from what the Fortran compiler would do. Usually, using the SEQUENCE attribute in the derived type definition (TYPE - END TYPE) can resolve most of these problems. In rare cases, the programmer needs to use non-standard structure components or structure definitions in order to define the structure properly.

**Character Strings.** Many OS API procedure interfaces are written in C and implicitly or explicitly expect that character strings are passed using the conventions of C. Fortran and C have very different ideas about how to pass character strings as procedure arguments. Fortran considers character strings to be fixed-length strings. C considers a character string to be a sequence of characters which is null-terminated. In such cases, most Fortran compilers offer a transformation function, usually named CARG() or CSTRING(), which takes a Fortran CHARACTER variable and passes it in the proper C fashion.

**External Names (Linking).** In many Unixes, there is a naming difference between the OS API name and what the Fortran compiler (or linker) produces. Very often a Fortran call to an OS API routine will add an underscore (_) character to the beginning or end of the procedure name when it is made available to the linker. Very often, the procedures in the OS API libraries will not have this extra underscore in their external names. This makes it virtually impossible to call a system routine (usually written in C) directly from Fortran (Lucio Chiapetti).

Following is a short summary of what is necessary in order to make OS API calls in various operating systems.

In HP's OpenVMS operating system, the OS API calls are provided through several API libraries.

The Run Time Library (LIB$) calls generally have a Fortran friendly user interface. The same is true for most of the utility library routines. The System Services Library (SYS$) calls generally have a lower-level user interface. The SYS$ calls make extensive use of unsigned integers, pass-by-value, and complex data structures. Compaq Fortran for OpenVMS has an extensive set of extensions which can overcome these problems. And in all of the API libraries, character strings are always passed by fixed-length string descriptor, the default method for passing character strings in HP Fortran for OpenVMS.

On Windows systems, the Windows API uses the C language calling conventions and implicitly assumes that the developer will be writing his/her application in C, C++, or C#. All of the PC-based Fortran compilers offer extensions which help the Fortran programmer make calls which are compatible with the Windows API. Most of the PC-based Fortran compilers offer an extension for pass-by-value. Character strings are a big problem, since the Win32 API expects all character strings to be passed by reference and to be null-terminated in the C fashion. Most Fortran compilers offer the CARG() and/or CSTRING() transformation functions. Either of these functions takes a regular Fortran character string and passes it in the proper C fashion. In addition, most Fortran compilers include documentation and code samples which show the user how to make the system calls.

In many Unixes, Fortran programmers have to resolve the difference in external names between what the compiler or linker produces and what the OS API routine name is. Some compilers have a compiler option that will remove the underscore. If this is not available, the programmer must write a jacket routine in C which has the underscore in its public name and which in turn calls the system routine that does not have the underscore (Lucio Chiapetti).

**Myth: Fortran can't perform database operations.**

Almost all database products include support for embedded database calls from the source code of all languages that are actively supported on the same platform as the database system. For example, the Oracle database products for HP's Open-

VMS operating system (regular Oracle and Oracle/Rdb) include support for embedded SQL calls not only from C but also from other popular languages, including Basic, COBOL, Pascal, and Fortran.

Usually, the developer uses these products by running the source code with the embedded database calls (usually, though not always in SQL) through a pre-compiler before using the regular compiler. The pre-compiler transforms the database statements into subroutine calls and produces an intermediate file, which is then fed into the regular compiler.

One advantage that Fortran has over languages such as C is that the Fortran model for character strings is usually the same as that used for character fields in most databases. In both Fortran and databases, character strings are fixed-length strings with blank-extended semantics and no magic characters. Both Fortran and databases also use the relational and assignment operators in the same way for character strings.

Canaima Software (http://www.canaimasoft.com), which has gone out of business, produced f90SQL, a Fortran 90 library that allows a developer to make SQL calls to the Microsoft Windows ODBC API. There are copies of the free introductory version, f90SQL-lite, available on the Internet.

I have no idea if full versions of f90SQL are available.

**Myth: All Fortran compilers are expensive.**

On some platforms, there are inexpensive Fortran compilers. Of course, the definition of "inexpensive" depends a great deal on the person making the evaluation. For the purpose of this article, I am defining "inexpensive" as a list price less than $300 before tax or shipping. This is not a lot of money. It is far less than a person would need to pay for renting an apartment for a month or buying a run-down used car.

Many vendors offer academic versions of their full-featured products for substantially less than the price they charge for commercial use. Usually, the buyer must be a bona-fide student or faculty member of an accredited school.

Following is a list of some inexpensive Fortran compilers for the Windows and Linux environments on Intel hardware, as of May 2011.

Absoft offers an academic version of its Pro Fortran compiler.

Lahey offers the Express version of LF95. It is a full-featured version of the LF95 compiler, without any limitations. LF95 Express includes the optimizing compiler, linker, command line debugger, librarian, support for creating DLLs and calling the Windows APIs, online documentation, and unlimited e-mail support. It does not include the IDE or any advanced tools.

Silverfrost offers a Personal Edition of FTN 95 for .NET. It is restricted to personal, noncommercial use. It is a complete, full-featured implementation of the Commercial Developer Edition.

There also are some free or very low-cost compilers available for subsets or older versions of the Fortran standard.

- The Fortran Company (http://www.fortran.com) offers the F subset compiler 3 either on CD or by download. You can get the download for free via anonymous FTP file transfer.

- The Free Software Foundation GNU Project (http://gcc.gnu.org/wiki/GFortran) offers gfortran, a free Fortran 95 compiler with many Fortran 2003 features. Another free Fortran 95 compiler is g95. The two compilers are developed by separate and independent teams.

- Some Linux distributions include a Fortran compiler as part of the package.

Earlier versions of Fortran compilers are also available on the secondary market. Auction sites such as eBay often list them.

**Myth: There are few programmers who know Fortran.**

Admittedly, Fortran is a lot less popular these days than its main rivals: C, C++, C#, and Java.

However, that does not mean that there are not a lot of programmers who are proficient in Fortran.

Although definite numbers are difficult to obtain, the number of Fortran programmers in the USA alone is in the hundreds of thousands at least.

Due to its long history, there are a lot of programmers who learned Fortran many years ago.

Most of these programmers are still professionally active. Many engineering programs still offer Fortran either as a required subject or as an elective.

**Myth: Fortran is no easier to learn than other popular programming languages such as C, C++, C#, or Java.**

Actually, Fortran is a lot easier to learn than are most other programming languages. Ease of use and ease of learning it were designed into the language right from the beginning ([IBM 1954, 2-3]).

Modern Fortran has a large number of features that make it easy to learn and a good choice for teaching proper programming practices.

- The grammar and syntax of Fortran is natural and straightforward for someone whose native language is English and who has a strong enough mathematical knowledge so that they are comfortable with high school algebra.

- Fortran does not have nearly as many irregularities of rules as do other languages. Once you have learned the way to use a feature in one case, the other cases almost always follow the pattern quite naturally. There are very few wicked surprises and gotchas.

- Most common features and operations are implemented in as simple and natural a manner as possible, from the application developer's point of view.

- Many commonly used language features, e.g., I/O operations, are implemented as elements of the language instead of functions.

- Many language features are implemented at a much higher level than is common in C or C++.

This allows the programmer to do in one statement what often takes 3 to 10 statements in other languages.

- Character variables are a first class data type. Thus, character operations are much easier and more straightforward.

- Input / Output (I/O) operations are much easier and more straightforward. The major I/O operations are implemented as statements, not as procedures.

- The use of interface blocks and modules allows the programmer to catch many procedure interface errors prior to execution time.

- Many common programming operations can be accomplished without the use of pointers. This makes it safer and easier to program.

- The availability of modules encourages modular, structured design and the use of modern software development practices.

From my own observations and those of others, it is reasonable to expect that a programmer can become competent in Fortran within one month and proficient within 6 months.

**Myth: It is hard to find Fortran expertise.**

Fortran expertise is neither easier nor harder to find than any other kind of software expertise.

The same methods of finding and evaluating people for other software specialties also apply equally well to finding Fortran programmers.

Much of this concern is based on the two previous myths. Belief in these myths often motivates hiring managers to fail to apply themselves as much as they would in finding people for other software specialties.

Other difficulties in finding good quality Fortran programmers result from pathologies that afflict either the software development process or the hiring and evaluation process generally. Many of these problems develop due to cultural problems within organizations.

Following are a few general observations about the hiring process. A detailed discussion is well beyond the scope of this article.

The culture in many organizations does not consider management to be a separate profession in its own right, with its own set of knowledge, skills, and abilities, and its own standards of excellence.

This means that persons who perform management functions, such as hiring and evaluation of people, are often chosen for their technical expertise in other fields, rather than for their business management aptitude or expertise. Often, the persons with hiring authority are not trained in proper personnel hiring and evaluation processes, so standard and accepted methods are not used when developing job descriptions or evaluating candidates.

Another problem is that the hiring process, whether for permanent employees or consultants, is a high risk proposition for the decision maker. First, it is quite expensive. In addition, in many firms, it is a no-win proposition. If the decision maker makes a good choice, he/she is simply doing his/her job. However, if he/she makes a hiring mistake, then the decision maker can forfeit future career advancement opportunities or even risk getting fired. Therefore, in many cases, there is a very strong incentive to over-specify on job qualifications and to be very stringent when screening candidates against the specified qualifications.

This risk/reward relationship for the individual decision maker is usually quite different from the risk/reward relationship for the firm that needs to have the work done. The firm may suffer substantial additional direct costs or miss significant market opportunities due to delays in obtaining qualified people to do the work. The importance of this discrepancy between the decision maker's risk profile and the firm's risk profile cannot be overstated.

Another obstacle to employing qualified expertise is that many firms and managers insist that all of the work be done on-site, even when the nature of the work requires little or no work actually to <#>.

be done on-site. This demand narrows the pool of candidates to those who are available within a reasonable commuting distance of the work site.

Insisting that the work be done on-site is seldom necessary for work that is nearly all software development. The high availability and low cost of fax, phone, e-mail, teleconferencing, and videoconferencing makes telecommuting practical in situations that do not require a lot of face-to-face contact. I personally have done two whole projects via telecommuting. In both cases, the customers were reluctant to approve telecommuting, but they agreed to the arrangement because they had no facilities available at the work place. I completed both projects in a very smooth manner and the customers were very pleased with the results.

**Myth: A program written in Fortran is unmaintainable.**

Concerns about the maintainability of software written in any language have several origins.

These include the software development practices of the developers, technical expertise of the developers, and availability of expertise after the initial project is completed.

Much of the ease or difficulty of maintaining a substantial application is due to the care that the original developers used in developing the application. Consistent adherence to recognized good practices of software development can make a program substantially easier to maintain. One of the best collections of good code writing practices are the recommendations in both editions of the award winning book *Code Complete: A Practical Handbook of Software Construction* by Steve McConnell ([McConnell 1993] and [McConnell 2004]). Most of these practices are equally applicable to development in Fortran, as they are to any other language. One word of caution. The author has a very low opinion of Fortran, but he bases his opinion on FORTRAN 77 and ignores later versions of the language in the first edition of his book. Unfortunately, he ignores Fortran completely in the second edition of his book, which is heavily oriented toward the Microsoft Visual Studio languages.

Concerns about the technical expertise of the developers or availability of expertise for maintenance and enhancement work is due largely to belief in the three preceding myths. As my answers to these myths show, such concerns are largely unfounded. Even in the unlikely event that a firm cannot find suitable Fortran expertise for a prospective project, it usually is fairly easy and inexpensive to train capable software developers in Fortran.

**Myth: Nobody believes these myths any longer.**

Anyone who believes this myth obviously does not live in the same world that I live in. As I mentioned in my introduction, I have personally heard each of these at least once. And, I should add, the people who proclaimed these myths did so with a straight face.

I prepared the original list in November 1998 as part of preparation for a talk on Fortran 90, 95, and Fortran 2003 to the local (Milwaukee, WI) chapter of the IEEE. As an introduction to the talk I presented the list as a True/False test on what people really know about Fortran. In spite of the fact that the audience was largely quite knowledgeable about Fortran, over half of the audience agreed that “Fortran is only for scientific and engineering applications” and “It is difficult to do character operations easily in Fortran”.

I have talked with a lot of people who have never actually programmed in Fortran. Yet, they “know” that all of these myths about Fortran are true- Usually, though not always, they learned this “knowledge” from their Computer Science professors.

For example, in 1998, I was talking with a young man who recently earned his BSCS degree.

He told me emphatically that character operations (in any language) were inherently difficult. His professors told him that. Of course, he had done all of his class assignments in C and C++. He had no direct experience with any other language.

I have had many other similar experiences. In most of the cases, the persons who “know” what Fortran is like are likely to have learned all that they know from one comparison course of several programming languages. In almost all of these cases, the version of Fortran that was presented was usually FORTRAN 66. Nothing about any later versions of Fortran.

**References.**

[ANSI 1966] American National Standards Institute (ANSI). American National Standard X3.9-1966, FORTRAN (FORTRAN 66).

[Freedman 1995] Freedman, Alan. *Computer Glossary*, 7 ed. (New York, NY: American Management Association, 1995).

[IBM 1954] IBM Corporation. “Specifications for the IBM Mathematical FORmula TRANslating System, FORTRAN”. (New York, NY: IBM Corporation, 1954).

[IEEE 1990] Institute of Electrical and Electronics Engineers (IEEE). *IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terms*.

[ISO 1991] International Standards Organization (ISO). ISO/IEC 1539-1:1991, *International Standard Programming Language Fortran (Fortran 90)*.

[ISO 1997] International Standards Organization (ISO). ISO/IEC 1539-1:1997, *International Standard Programming Language Fortran (Fortran 95)*.

[ISO 2004] International Standards Organization (ISO). ISO/IEC 1539-1:2004, *International Standard Programming Language Fortran (Fortran 2003)*.

[ISO 2010] International Standards Organization (ISO). ISO/IEC 1539-1:2010, *International Standard Programming Language Fortran (Fortran 2008)*.

[McConnell 1993] McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction,* 1 ed. (Redmond, WA: Microsoft Press, 1993). [McConnell 2004] McConnell, Steve. *Code Complete: A Practical Handbook of Software Construction*, 2 ed. (Redmond, WA: Microsoft Press, 2004). [Reid 2003] Reid, John. "The New Features of Fortran 2003". ISO/IEC JTC1/SC22/WG5 N1648.

[Reid 2008] Reid, John. "The New Features of Fortran 2008". ISO/IEC JTC1/SC22/WG5 N1729.

## Contributors.

Craig Dedo, the author of this article, is a computer consultant specializing in programming and software development. He has written programs in a variety of application areas, including engineering, accounting, manufacturing, and shop floor control, primarily under the OpenVMS and Windows operating systems. Craig has used Fortran extensively for over 30 years. From 1994 to 2003, he was a voting member of J3, now PL22.3, the ISO/ANSI Fortran Standards Committee. He is active in the Milwaukee Software Process Improvement Network and the local chapter of the Project Management Institute (PMI). He holds a B.S. degree in Applied Mathematics, Engineering, and Physics from the University of Wisconsin – Madison, an MBA degree from the University of Wisconsin – Milwaukee, and a Master of Project Management degree from the Keller Graduate School of Management of DeVry University in Chicago. He lives in Brookfield, WI.

Many thanks to all of the other contributors: Martin Ambuhl, Mark Brown, Lucio Chiapetti, James Giles, Ken Hamilton, Dick Hendrickson, Richard Maine, Michael Metcalf, Toon Moene, Dan Nagle, Chris Nahr, Clive Page, Ken Plotkin, Tim Prince, Mike Ross, Gordon Sande, Gary Scott, Ron Shepard, and Jonathon Stott.