Fortran Returns

Shimon Panfil,Ph. D.
Industrial Physics and Simulations
http://industrialphys.com

March 5, 2010

1 Fortran History

1.1 Prehistory, before 1970

FORTRAN (**For**mula **Tran**slating) is the first ever created programming language [1]. In late 1953, John W. Backus submitted a proposal to his superiors at IBM to develop a more practical alternative to assembly language for programming their IBM 704 mainframe computer. The first manual for FORTRAN appeared in October 1956, with the first FORTRAN compiler delivered in April 1957. This was an optimizing compiler, because customers were reluctant to use a high-level programming language unless its compiler could generate code whose performance was comparable to that of hand-coded assembly language.

The language was widely adopted by scientists for writing numerically intensive programs, which encouraged compiler writers to produce compilers that could generate faster and more efficient code. The inclusion of a complex number data type in the language made FORTRAN especially suited to technical applications such as electrical engineering. The increasing popularity of FORTRAN spurred competing computer manufacturers to provide FORTRAN compilers for their machines, so that by 1963 over 40 FORTRAN compilers existed. For these reasons, FORTRAN is considered to be the first widely used programming language supported across a variety of computer architectures.

New era of portable software has started. FORTRAN was a practical solution it was not absolutely machine-independent and has few other deficiencies. An attempt to produce more theoretically sound and clean programming language yielded ALGOL (**Algo**rithmic **L**anguage) in 1958 [2]. ALGOL greatly influenced many other languages and became the de facto way algorithms were described in textbooks and academic works for almost the next 30 years. It was implemented for a number of computers but has never got as popular as FORTRAN for professional programming. In university (the beginning of 70s) I was taught ALGOL in basic programming course, but more advanced courses used FORTRAN.

LISP was invented by John McCarthy in 1958 [3]. LISP was originally created as a practical mathematical notation for computer programs and quickly became the favored programming language for artificial intelligence (AI) research. As one of the earliest programming languages, LISP pioneered many ideas in computer science, including tree data structures, automatic storage management, dynamic typing, and the self-hosting compiler. LISP was first implemented by Steve Russell on an IBM 704 computer. Russell had read McCarthy's paper, and realized (**to McCarthy's surprise!**) that the LISP eval function could be implemented in machine code. Note the difference: FORTRAN was invented to make practical programming more efficient, LISP was implemented on the same machine at the same time to the surprise of its author.[1]

To complete the description I must mention COBOL (**Co**mmon **B**isness **O**riented **L**anguage) [4] which appeared in 1959 and became the most popular programming language. In 1997, the Gartner Group reported that 80% of the world's business ran on COBOL with over 200 billion lines of code in existence and with an estimated 5 billion lines of new code annually.

So prehistoric programmers were divided into three groups:

Business The largest group using COBOL;

Numerical Scientists and engineers using FORTRAN for computation and sometimes ALGOL for publications;

Theoretics Small but fast growing and active group using all other languages.

I shall discuss only numerical programming below, I myself belong to this group and have no professional knowledge in other kinds of programming.

Current state of FORTRAN was defined by the first standard became known as FORTRAN 66 (although many continued to refer to it as FORTRAN IV, the language upon which the standard was largely based). FORTRAN 66 effectively became the first "industry-standard" version of FORTRAN. FORTRAN 66 included:

- Main program, SUBROUTINE, FUNCTION, and BLOCK DATA program units
- INTEGER, REAL, DOUBLE PRECISION, COMPLEX, and LOGICAL data types
- Intrinsic and EXTERNAL (e.g., library) functions
- Assignment statement
- GOTO, assigned GOTO, and computed GOTO statements
- Logical IF and arithmetic (three-way) IF statements
- DO loops
- Identifiers of up to six characters in length
- Comment lines

Note that operating systems are not invented yet. FORTRAN 66 is not merely compiler but "Programming system" comprising all means needed to use computer.

[1]Cf. Dijkstra: "Computer science is no more about computers than astronomy is about telescopes." The design and deployment of computers and computer systems is generally considered the province of disciplines other than computer science [5].

1.2 Ancient Times, 1970s-1980s

After the release of the FORTRAN 66 standard, compiler vendors introduced a number of extensions to "Standard Fortran", prompting ANSI in 1969 to begin work on revising the 1966 standard. Final drafts of this revised standard circulated in 1977, leading to formal approval of the new FORTRAN standard in April 1978. The new standard, known as FORTRAN 77, added a number of significant features to address many of the shortcomings of FORTRAN 66:

- Block IF and END IF statements, with optional ELSE and ELSE IF clauses, to provide improved language support for structured programming
- DO loop extensions, including parameter expressions, negative increments, and zero trip counts
- OPEN, CLOSE, and INQUIRE statements for improved I/O capability
- Direct-access file I/O
- CHARACTER data type, with vastly expanded facilities for character input and output and processing of character-based data
- PARAMETER statement for specifying constants
- SAVE statement for persistent local variables

In this revision of the standard, a number of features were removed or altered in a manner that might invalidate previously standard-conforming programs. (Removal was the only allowable alternative to at that time, since the concept of "deprecation" was not yet available for ANSI standards.)

Operating systems came to exist which take many functions from "programming systems" leaving them merely compiler and libraries. New programming languages appear, the most notable were UNIX and C. FORTRAN 77 became old fashioned but still preferred language for numerical programs ([6],[7]).

1.3 Medieval Period, 1980-2005

Mini-computers, micro-computers, personal computers, multimedia, internet, Windows ...! Intensive growth of non-numerical applications on one hand and development of "user-friendly" numerical software like Matlab, Mathematica, Maple on the other hand have overshadowed numerical programming. (Scientific or numerical problem? It is Matlab!)

The numerical programming community is still active but bound to large computer centers in industry, national laboratories and universities. High Performance Computer vendors provide efficient Fortran compilers for their machines and people working on new Fortran standards [8]. In particular Fortran 90 standard includes all features necessary for modern descent language:

- Free-form source input, also with lowercase Fortran keywords
- Identifiers up to 31 characters in length
- Inline comments

- Ability to operate on arrays (or array sections) as a whole, thus greatly simplifying math and engineering computations.
- RECURSIVE procedures
- Modules, to group related procedures and data together, and make them available to other program units, including the capability to limit the accessibility to only specific parts of the module.
- A vastly improved argument-passing mechanism, allowing interfaces to be checked at compile time
- User-written interfaces for generic procedures
- Operator overloading
- Derived/abstract data types
- New data type declaration syntax, to specify the data type and other attributes of variables
- Dynamic memory allocation by means of the ALLOCATABLE attribute and the ALLOCATE and DEALLOCATE statements
- POINTER attribute, pointer assignment, and NULLIFY statement to facilitate the creation and manipulation of dynamic data structures
- Structured looping constructs, with an END DO statement for loop termination, and EXIT and CYCLE statements for "breaking out" of normal DO loop iterations in an orderly way
- SELECT . . . CASE construct for multi-way selection
- Portable specification of numerical precision under the user's control [2]
- New and enhanced intrinsic procedures.

Fortran 95 was a minor revision, mostly to resolve some outstanding issues from the Fortran 90 standard. Nevertheless, Fortran 95 also added a number of extensions, notably from the High Performance Fortran specification:

- FORALL and nested WHERE constructs to aid vectorization
- User-defined PURE and ELEMENTAL procedures
- Pointer initialization and structure default initialization.

A number of intrinsic functions were extended (for example a dim argument was added to the maxloc intrinsic).

Several features noted in Fortran 90 to be deprecated were removed from Fortran 95.

F (programming language)[9] was designed to be a clean subset of Fortran 95 that attempted to remove the redundant, unstructured, and deprecated features of Fortran, such as the EQUIVALENCE statement. F retains the array features added in Fortran 90, and removes control statements that were obsoleted by

[2]Absolutely senseless and misleading feature

structured programming constructs added to both Fortran 77 and Fortran 90. F is described by its creators as "a compiled, structured, array programming language especially well suited to education and scientific computing."

However these attempts have (how to put it mildly?) only restricted success. Why? Because of a number of reasons:

1. Most of most of modern computer professionals including lecturers and book authors in programming know nothing about numerical programming and their recommendations on programming style and methodology are in many cases counterproductive when applied to this field;

2. Modern programming languages like C, C++, Java,... were designed primarily for non-numerical applications and are not efficient when applied to numerical programming;

3. Battle tested legacy code is written in old (FORTRAN 77 mainly) standard, so modern Fortran is not enough, one must learn also old one, but in this case why bother to new 2 languages, so even newcomers are tempted to stay with FORTRAN 77;

4. Even modern Fortran is at odds with operating systems;

5. There is no open source modern Fortran compiler for Linux, which prevents use of modern Fortran on Linux clusters (well known poor man supercomputer).

1.4 Our Times

New standard Fortran 2003 adds the following features:

- Derived type enhancements
- Object-oriented programming support: type extension and inheritance, polymorphism, dynamic type allocation, and type-bound procedures.
- Data manipulation enhancements: VOLATILE attribute, pointer enhancements, extended initialization expressions, and enhanced intrinsic procedures.
- Input/output enhancements: asynchronous transfer, stream access ...
- Procedure pointers.
- Support for IEEE floating-point arithmetic and floating point exception handling
- Interoperability with the C programming language.
- Support for international usage
- Enhanced integration with the host operating system

Efforts are underway to develop a revision to Fortran 2003, tentatively called Fortran 2008. As with Fortran 95, this is intended to be a minor upgrade, incorporating clarifications and corrections to Fortran 2003.

However the most important development is that Gnu Compiler Collection now includes Fortran 2003! Before 2005, gcc included g77, which implements FORTRAN 77 standard with some extensions. Development of the replacement was started in 2000 (g95 project [10]). In 2003 gfortran [11] forked from g95 and became a part of gcc from version 4.0. It includes support for the Fortran 95 language and is compatible with most language extensions supported by g77, allowing it to serve as a drop-in replacement in many cases. Parts of Fortran 2003 and Fortran 2008 have also been implemented. A number of commercial compilers for Windows and Linux also came to exist.

Modern Fortran standard is typical "designed by committee"[3] product. The defining characteristics of "design by committee" are needless complexity, internal inconsistency, logical flaws, banality, and the lack of a unifying vision [12].

Relation between FORTRAN 77 and Fortran 2003 resembles that between C and C++. Both C and FORTRAN 77 are small and easy to learn (though both have some strange features historically motivated) while both C++ and Fortran 2003 are large and bloated (however they improved their predecessors in a way). Specifically the most important improvement that C++ makes over C — that it made many more jobs for programmers is beyond the scope of the article, and second one — double slash comment was present in C as gcc extension for years. Fortran 2003 improvements are:

1. Dynamic memory allocation;
2. Pointers (different form C pointers!);
3. C and OS interoperability;
4. derived types (similar to C-structures);
5. modules;
6. recursive procedures;
7. whole array operations.

The most essential deprovement is kind parameter for real numbers: instead of usual single and double precision, modern Fortran allows you to define formally real(kind) to get the accuracy and range as you want. "Portable specification of numerical precision under the user's control" as it is described. However this is misleading because system will silently replace your definition by standard one or return error if none of standard types: 4 byte float, 8 byte double, 10 byte (x86-64) or 16 byte long double — can be used.

2 Fortran 2003 for C programmers

Most of modern Fortran constructions which may look strange for FORTRAN 77 programmers are quite natural for C-programmers (e.g. using pointers and

[3]"A camel is a horse designed by committee"

structures for building liked lists etc), though care is needed due to some differences (see Fortran language description e.g. [13],[14], [15] and [16]). For example though it may seem strange for C programmer, this program works:

```
program saveprog
    call s()
    call s()
    call s()
contains
    subroutine s()
        integer::count=0
        count=count+1
        print *,"execution",count
    end subroutine s
end program saveprog
```

Explanation is that every initialized variable gets save attribute automatically.

Below I will describe few modern Fortran features which are most useful in filling the gap between FORTRAN 77 tradition of scientific programming and modern languages and computers.

2.1 Modules

Modules allow to keep together data and functions logically connected. Fortran modules combine C include files with precompiled libraries. Actually modules are the preferred way to build libraries (not only in Fortran). Let us look at the example — well known random number generator:

```
module marsaglia
implicit none
private
public :: kiss, kisset
   INTEGER :: x=123456789, y=362436069, z=521288629, w=916191069
contains
   FUNCTION kiss ()
      integer :: kiss
      x = 69069 * x + 1327217885
      y = m (m (m (y, 13), - 17), 5)
      z = 18000 * iand (z, 65535) + ishft (z, - 16)
      w = 30903 * iand (w, 65535) + ishft (w, - 16)
      kiss = x + y + ishft (z, 16) + w
   contains
      function m(k, n)
         integer :: m, k, n
         m = ieor (k, ishft (k, n) )
      end function m
   END FUNCTION kiss
   function kisset (ix, iy, iz, iw)
      integer :: kisset, ix, iy, iz, iw
      x = ix
      y = iy
      z = iz
      w = iw
      kisset = 1
   end function kisset
end module marsaglia
```

The program or another model should issue ”use” statement, e.g.

```
PROGRAM example
   use marsaglia
   PRINT *, kiss ()
   PRINT *, kisset (1, 2, 3, 4)
   PRINT *, kiss ()
END PROGRAM example
```

Actually there is much more to say about the modules (you need not use all the members of module, you can rename members etc.) however I hope that this example demonstrates the way of gradual rewriting and improving of legacy code. You start from modules which simply call old functions, and then add members which improve the old ones.

2.2 Arrays

Arrays in modern Fortran are first class citizens and they are multidimensional (up to 7) as in FORTRAN 77, but they also can be dynamically allocated as in C. Note that allocated arrays are not pointers.

```
program mm3
    integer :: n,cac,i,j,k
    real,allocatable, dimension(:,:) :: a,b,c1,c2
    CHARACTER(len=32) :: arg
    real::start_time,stop_time
    real, parameter:: eps=1.0e-15
    cac=command_argument_count()
    if(cac/=1) then
        print *,"need 1 argument: dimension"
        stop
    end if
    CALL get_command_argument(1, arg)
    read (arg,*) n
    print *,"dimension=",n
    call cpu_time(start_time)
    allocate(a(n,n),b(n,n),c1(n,n),c2(n,n))
    call random_seed()
    call random_number(a)
    call random_number(b)
    call cpu_time(stop_time)
    print *,"matrix preparation",stop_time-start_time,"s"
    call cpu_time(start_time)
    c1=matmul(a,b)
    call cpu_time(stop_time)
    print *,"matmul",stop_time-start_time,"s"
    call cpu_time(start_time)
    c2=0.0
    do i=1,n
        do j=1,n
            do k=1,n
                c2(i,j)=c2(i,j)+a(i,k)*b(k,j)
            end do
        end do
    end do
    call cpu_time(stop_time)
    print *,"loop ijk",stop_time-start_time,"s"
    if(any(abs(c1-c2)>eps)) then
        print *,"different values, stop"
        stop
    end if
end program mm3
```

Output of that program shows clearly that good algorithm (in this case proper order of loops) is much more important than compiler optimization:

```
 dimension=        2000
 matrix preparation  0.16998800000000000      s
 matmul   13.462455000000000      s
 loop ijk   114.15922400000001      s
 loop ikj   280.43838099999999      s
 loop jik   116.78571899999997      s
 loop jki   15.608983000000080      s
 loop kji   22.701852999999915      s
 loop kij   280.77169299999991      s
```

2.3 Pointers

The amount of senseless discussions about pointers can be compared this that about goto [4]. The sentence like "Pointers are dangerous ..." have no more content than "Use of integers is extremely dangerous, while real numbers mostly obey the rules of mathematics, integers do not:$1/2*2 = 0$ use integers only when you absolutely need to!" or "Use of real numbers is extremely dangerous, they do not obey rules of mathematics:$pow(10.0, 300) - pow(10.0, 300) + 1 = 1.0$ while $pow(10.0, 300) + 1 - pow(10.0, 300) = 0.0$ use real numbers only when you absolutely need to!".

Actually neither pointers nor goto is evil. Unreasonable usage of any construction lead to poorly written programs and so does unreasonable prohibition. Pointers in Fortran are not merely addresses, they contain plenty of information about the object they point to. There is no pointer arithmetics. One can use them for building lists, trees and all that exactly as in C. However the main convenience of using pointers is demonstrated by the example — it is clear and compact representation of expressions, which helps to write complicated equations without errors.

[4]Compare "Letter O considered harmful" [1]

```
program heat_transfer
    implicit none
    integer ::m,n,c
    CHARACTER(len=32) :: arg
    real ::coeff
    real, allocatable, dimension(:,:), target :: plate
    real, allocatable, dimension(:,:) :: temp
    real, pointer, dimension(:,:) :: north, east, south, west, inside
    real, parameter :: tolerance =1.0e-4
    real :: diff
    integer ::j,niter
    c=command_argument_count()
    if(c/=2) then
        print *,"need 2 arguments"
        stop
    end if
    CALL get_command_argument(1, arg)
    read (arg,'(I10)') m
    CALL get_command_argument(2, arg)
    read (arg,'(I10)') n
    allocate(plate(m,n))
    allocate(temp(m-2,n-2))
 !initial conditions
    plate=0.0
    !boundary conditions
    plate(1:m,1)=1.0
    coeff=1.0/n
    plate(1,1:n)=[(coeff*j,j=n,1,-1)]
    inside=>plate(2:m-1,2:n-1)
    north=>plate(1:m-2,2:n-1)
    south=>plate(3:m,2:n-1)
    east=>plate(2:m-1,1:n-2)
    west=>plate(2:m-1,3:n)
    niter=0
    do
        temp=0.25*(north+east+south+west)
        diff=maxval(abs(temp-inside))
        niter=niter+1
        inside=temp
        if(diff<tolerance) then
            exit
        endif
    end do
    print *,plate(m/2,:)
end program heat_transfer
```

References

[1] http://en.wikipedia.org/wiki/Fortran

[2] http://en.wikipedia.org/wiki/ALGOL

[3] http://en.wikipedia.org/wiki/Lisp_(programming_language)

[4] http://en.wikipedia.org/wiki/COBOL

[5] http://en.wikipedia.org/wiki/Computer_science

[6] M. Kupferschmid. Classical Fortran. Marcel Dekker, 2002

[7] Clive G. Page. Professional Programmer's Guide to Fortran77, 2005

[8] T. M. R. Ellis, Ivor R. Philips, and Thomas M. Lahey. Fortran 90 programming. Addison-Wesley, 1994

[9] T. M. R. Ellis and Ivor R. Philips. Programming in F. Addison-Wesley, 1998

[10] http://www.g95.org/

[11] http://www.gfortran.org/

[12] http://en.wikipedia.org/wiki/Design_by_committee

[13] Stephen J. Chapman Fortran 95/2003 for Scientists and Engineers. McGraw-Hill, 2007

[14] Jeanne C. Adams, Walter S. Brainerd, Richard A. Hendrickson, Richard E. Maine, Jeanne T. Martin, Brian T. Smith. The Fortran 2003 Handbook, Springer 2009

[15] Walter S. Brainerd. Guide to Fortran 2003 Programming. Springer 2009

[16] Ian Chivers, Jane Sleightholme. Introduction to Programming with Fortran. Springer 2006