Computer Science in Ocean and Climate Research

Lecture 3: The Fortran Programming Language

Prof. Dr. Thomas Slawig

CAU Kiel
Dep. of Computer Science

Summer 2020

Contents

Contents

What is Fortran

- **For**mula **Tran**slation
- First higher programming language
- Originally procedural language
- Enhanced with object-oriented features
- Newest version: fully object-oriented

Tony Hoare, developer of Quicksort:

"I don't know what the language of the year 2000 will look like, but it will be called Fortran."

(Tony Hoare, taken from: Chivers, Sleightholme: Introduction to Programming with Fortran)

Why do we look at Fortran in this course

- Most climate models are written in Fortran...
- ...and also many simulations programs for technical and other scientific applications.
- Fortran Compiler is one of the top 10 algorithms of the 20th century (Society of Industrial and Applied Math. 2000)

from SIAM News, Volume 33, No. 4

The Best of the 20th Century: Editors Name to 10 Algorithms

by Barry A. Cipra

- For long time, Fortran was the language with the best compilers, i.e., the ones that generated the fastest code (then C became similarly fast).
- Many old codes (legacy codes) are written in Fortran. Often, a complete new implementation is too much effort.

Many numerically complex codes are still in Fortran

Standard Performance Evaluation Corporation

benchmarks | results | contact | site map | site search | help

sults
h
y
n
verview
ents
Rules
t
ks
chmarks
cations
cations
ks
06
Links

CFP2006 (Floating Point Component of SPEC CPU2006):

Benchmark	Language	Application Area	Brief Description
410.bwaves	Fortran	Fluid Dynamics	Computes 3D transonic transient laminar viscous flow.
416.gamess	Fortran	Quantum Chemistry.	Gamess implements a wide range of quantum chemical computations. For the SPEC workload, self-consistent field calculations are performed using the Restricted Hartree Fock method, Restricted open-shell Hartree-Fock, and Multi-Configuration Self-Consistent Field
433.milc	C	Physics / Quantum Chromodynamics	A gauge field generating program for lattice gauge theory programs with dynamical quarks.
434.zeusmp	Fortran	Physics / CFD	ZEUS-MP is a computational fluid dynamics code developed at the Laboratory for Computational Astrophysics (NCSA, University of Illinois at Urbana-Champaign) for the simulation of astrophysical phenomena.
435.gromacs	C, Fortran	Biochemistry / Molecular Dynamics	Molecular dynamics, i.e. simulate Newtonian equations of motion for hundreds to millions of particles. The test case simulates protein Lysozyme in a solution.
436.cactusADM	C, Fortran	Physics / General Relativity	Solves the Einstein evolution equations using a staggered-leapfrog numerical method
437.leslie3d	Fortran	Fluid Dynamics	Computati (CFD) usir with Linear the MacCo time integr
444.namd	C++	Biology / Molecular Dynamics	Simulates systems. T atoms of a
447.dealII	C++	Finite Element Analysis	deal.II is a targeted at and error e solves a H with non-c
450.soplex	C++	Linear Programming, Optimization	Solves a li simplex al algebra. T planning a
453.povray	C++	Image Ray-tracing	Image rend 1280x1024 landscape objects wit noise funct
454.calculix	C, Fortran	Structural Mechanics	Finite elem nonlinear 3 Uses the S
459.GemsFDTD	Fortran	Computational Electromagnetics	Solves the using the fi domain (FI
465.tonto	Fortran	Quantum Chemistry	An open so package, u design in F places a co Hartree-Fo calculation experimen
470.lbm	C	Fluid Dynamics	Implement

How does a Fortran program look like

- Older programs are purely procedural
- Newer programs have object-oriented elements
- Because of the long history of the language there exist different standards ...
- ... and also different code formats (fixed, free format, s.u.)

For what purpose Fortran is useful?

- High Performance Computing (HPC)
- Computationally expensive numerical programs
- Simulations
- Many numerical libraries are written in Fortran
- Generally: everything that can be done also in C
- Coupling with C, C++, Java, Python is possible

Contents

Fortran history

- Developed 1954-57 at IBM under John Backus (see Backus-Naur form, BNF, method to describe programming languages).
- Idea: enable programming of formulas in an effective way, without knowledge in hardware but also without performance losses.
- Proximity to hardware and efficiency more important than abstract concept.
- Replacement for Assembler programming for complex programs
- 1966: Standard Fortran 66 ASA (American Standards Association, predecessor of ANSI)

John Backus
1924-2007

Picture: Pierre Lescanne GNU FDL

Fortran history

from SIAM News, Volume 33, No. 4 (2000)

The Best of the 20th Century: Editors Name to 10 Algorithms

by Barry A. Cipra

- 1957: John Backus leads a term at IBM in developing the Fortran optimizing compiler.
  The creation of Fortran my rank as a single important event in the history of computer programming: Finally, scientists (and others) could tell the computer what they wanted it to do, without having to descend into the netherworld of machine code. Although modest by modern compiler standards - Fortran I consisted of a mere 23,500 assembly-languages instructions - the early compiler was nonetheless capable of surprisingly sophisticated computations. As Backus himself recalls in a recent history of Fortran I, II and III, published in 1998 in the *IEEE Annals of the History of Computing*, the compiler "produced code of such efficiency that its output would startle the programmers who studied it."

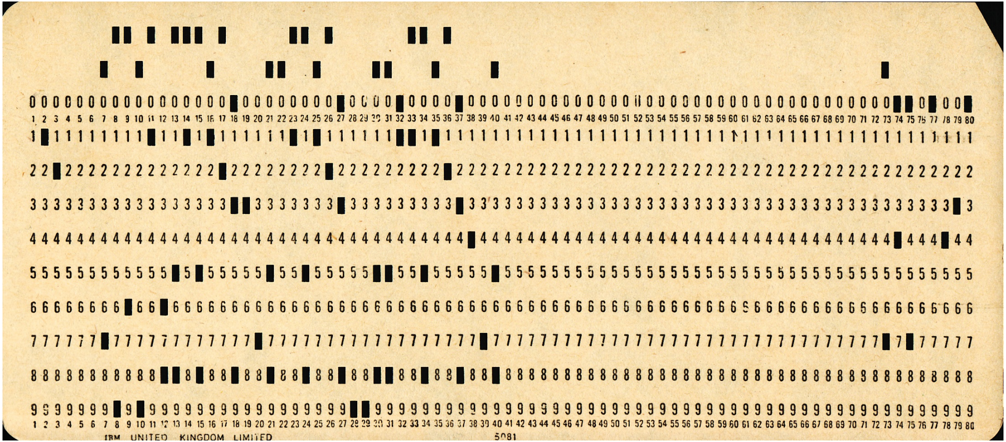
Fortran versions

- 1957: Fortran I- IV, 1966: Fortran 66
- 1978: Fortran 77: still widely used, only fixed code format (fixed format)
- 1990: Fortran 90: New: free code format (free format), dynamic memory,advanced control structures (do-while),pointer, recursion, abstract data types, array operations (A=B). Some features of object orientation (modules containing data types and functions, similar to classes)
- 1996: Fortran 95: "cleaning", some redundant features are omitted, F95 compiler were faster.
- 2004: Fortran 2003: More object-oriented features (inheritance)
- 2010: Fortran 2008: Parallelization, coupling with C
- 2018: Fortran 2018: Smaller update

Contents

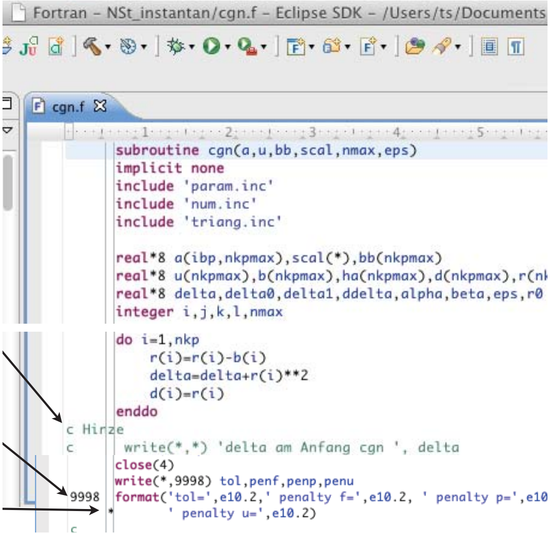
Fixed format

- Fixed format (standard until F77): motivation by punch cards



Pete Birkinshaw from Manchester, UK / CC BY (https://creativecommons.org/licenses/by/2.0)

Fixed format



Special meaning of columns:

- 1: commentary
- 2-5: labels
- 6: continuation mark (concatenation of two lines)

Since Fortran 90: Free format

- Most of the restrictions of F77 are omitted
- Continuation mark now at the end of the line
- Commentary starting with "!" to the end of the line.

```
!
!   bgc.f90
!   tmmbgc
!
!   Model for PO4 (phosphate, y(:,0)) and DOP (dissolved organic phosphorus, y(:,1)
!   - assuming ybgc is zero
!   - conforms to BGC_1D_API
!
!   Created by Jaroslaw Piwonski (CAU) on 10.02.10.
!   Copyright 2010 CAU. All rights reserved.
!

subroutine bgc(t, dt, lat, insol, dayfrac, ice, wind, ntracer, nlayer, h, d, salt, t
    implicit none
    ! input variables
    real*8  :: t, dt, lat, insol, dayfrac, ice, wind
    integer :: ntracer, nlayer, nparam
    real*8  :: h(0:nlayer-1), d(0:nlayer-1), salt(0:nlayer-1), temp(0:nlayer-1)
    real*8  :: y(0:nlayer-1, 0:ntracer-1)
    real*8  :: ybgc(0:nlayer-1, 0:ntracer-1)
    real*8  :: u(0:nparam-1)
    ! integer value parameters
    integer :: nbiostep     = 8
    integer :: jeuphotic    = 2
    ! real value parameters
    real*8  :: lambda                       ! remineralization rate               : 1 /
    real*8  :: alpha                        ! maximum export rate                 : uM
    real*8  :: nu                           ! fraction of DOP
    real*8  :: K_PO4                        ! half saturation constant: PO4       : uM
```

Contents

Some special features of Fortran

- Case-insensitive
- Main program starts with keyword *program*
- Modularization: *subroutine*, *function*
- Data types: *integer*, *real,double precision (=real*8)*
- Arrays

ad_dtoc.f

```fortran
      program dtoc

      implicit none

c 080301
c
c Instantane Kontrolle mit Hilfe von AD
c
c Programm berechnet discrete time (sub)optimal control fuer NSE(t)
c auf der Basis einer diffusiv-impliziten Zeitdiskretisierung erster
c Ordnung. Dabei wird in jedem Zeitschritt ein Kontrollproblem fuer
```

```fortran
subroutine cgp(a,u,bb,scal,nmax,eps)

include 'param.inc'
include 'num.inc'
include 'nump.inc'
include 'triang.inc'

real*8 a(ibp,nkmax),scal(*),bb(nkpmax)
real*8 u(nkpmax),b(nkpmax),ha(nkpmax),d(nkpmax),r(nkpmax)



delta=0.
nmax=1
do i=1,nkp
     b(i)=bb(i)/scal(i)
     u(i)=u(i)*scal(i)
enddo
```

Some special features of Fortran

- Implicit declaration of variables: variables that start with *i, j, k, l, m, n* are *integers*,
- … others are *real*
- *implicit none* statement disables this feature and is recommended.
- Typical loop: *do … enddo*

ad_dtoc.f

```
      program dtoc

      implicit none

c 080301
c
      subroutine cgp(a,u,bb,scal,nmax,eps)

      include 'param.inc'
      include 'num.inc'
      include 'nump.inc'
      include 'triang.inc'

      real*8 a(ibp,nkmax),scal(*),bb(nkpmax)
      real*8 u(nkpmax),b(nkpmax),ha(nkpmax),d(nkpmax),r(nkpmax)

      delta=0.
      nmax=1
      do i=1,nkp
          b(i)=bb(i)/scal(i)
          u(i)=u(i)*scal(i)
      enddo
```

loop counter variable is not declared

end statement

- Old versions: no end statement for loops
- Jump labels and continues statement were used instead
- *continue* statement does not do anything
- ... was replaced by *enddo* statement

```
      do 100 i=1,100
c         do something
  100 continue
```

old: < F77

```
      do i=1,100
c         do something
      enddo
```

new: since F77

Even worse . . .

- Whitespaces in names are ignored
- Together with the implicit declaration and the usage of the *continue* statement (instead of an *end* statement) there could be problems.
- Left-hand picture: correct loop
- : Right-hand picture: typo: assignment to *real* variable *do100i* and no loop:

```fortran
      do 100 i=1,100
c         do something
  100 continue
```

```fortran
      do 100 i=1.100
:         do something
  100 continue
```

- Famous Fortran bug that reportedly caused the crash of a rocket, but is now denoted as legend.

Some special features of Fortran

Global variables in *common* blocks:

- In every subprogram where the common block with the same name is used, the included variables are shared, ...
- ... even if they had different names in the respective subprograms.
- A common block represents a connected area of memory, having the length of the variables that are assigned to it,
- ... but types and names may be different in every subprogram, as long as the total memory in the common block remains the same.
- Error-prone!
- Can be found in may old programs.

*ex1.f

```
      subroutine bla1
      common/cb1/x
      x = 1.0
      end

      subroutine bla2
      common/cb1/y
      y = y+1.0
      end
```

Contents

First improvements . . . Fortran 90

- Modules instead of common blocks
- ... contain abstract data types and corresponding functions
- similar to C structs
- first step to object-orientation
- declarations with ::
- dynamically allocated memory: *allocate, deallocate*
- *do-while* loops

```
MODULE bla_module
    TYPE bla
        REAL :: x
        INTEGER :: y
    END TYPE bla

    CONTAINS

    SUBROUTINE blub

    ! do something...

    END SUBROUTINE blub

end bla_module
```

In F77: *gotos* and *arithmetic if*

Arithmetic IF

From Wikipedia, the free encyclopedia

The **arithmetic IF** statement has been for several decades a three-way arithmetic conditional statement, starting from the very early version (1957) of Fortran, and including FORTRAN IV, FORTRAN 66 and FORTRAN 77. Unlike the logical IF statements seen in other languages, the Fortran statement defines three different branches depending on whether the result of an expression was negative, zero, or positive, in said order, written as:

```
IF (expression) negative,zero,positive
```

While it was originally the only kind of IF statement provided in Fortran, the feature was used less and less frequently after the more powerful logical IF statements were introduced, and was finally labeled obsolescent in Fortran 90.

```
      XES=0.0
      XEC=0.0
      DO 4 I=1,NEF
      ARG=BE(I)*T+CE(I)
      XES=XES+AE(I)*SIN(ARG)
      XEC=XEC+AE(I)*COS(ARG)
    4 CONTINUE
      ECC=SQRT(XES*XES+XEC*XEC)
      TRA=ABS(XEC)
      IF(TRA.LE.1.0E-08) GO TO 10
      RP=ATAN(XES/XEC)
      IF(XEC) 11,10,12
   11 RP=RP+PI
      GO TO 13
   12 IF(XES) 14,13,13
   14 RP=RP+2.0*PI
      GO TO 13
   10 IF(XES) 15,16,17
   15 RP=1.5*PI
      GO TO 13
   16 RP=0.0
      GO TO 13
   17 RP=PI/2.0
   13 PERH=RP/PIR
```

Fortran uses *call-by-reference* only . . .

Exchange values of two variables

ex1.f90

```
program ex1
    implicit none
    integer :: x,y
    x = 1
    y = 2
    write(*,*) "vorher:  x= ", x, "y = ", y
    call swap(x,y)
    write(*,*) "nachher: x= ", x, "y = ", y
end program ex1

subroutine swap(x,y)
    integer :: x,y,help
    help = x
    x = y
    y = help
end subroutine swap
```

```
m-114:CallByReference ts$ gfortran ex1.f90
m-114:CallByReference ts$ ./a.out
 vorher:  x=           1 y =            2
 nachher: x=           2 y =            1
```

Different in C:

ex1.c

```c
#include <stdio.h>

void swap(int x,int y) {
    int help=x;
    x = y;
    y = help;
}

int main() {
    int x=1,y=2;
    printf("x = %i, y = %i\n",x,y);
    swap(x,y);
    printf("x = %i, y = %i\n",x,y);
    return 0;
}
```

```
m-114:CallByReference_C ts$ gcc ex1.c
m-114:CallByReference_C ts$ ./a.out
x = 1, y = 2
x = 1, y = 2
```

Fortran 90 and 95

- Array operations possible
- Similar to MATLAB®/Octave
- F95: additional *where* statement
- Built-in linear algebra routines (e.g., matrix multiplication).

```
real :: A(10,10)
real :: B(10,10)

A = B
A = A + B

where (A<0.0)
    A = 0.0
elsewhere
    ! ....
end
```

Fortran 2003/2008: Object-oriented programming

- Fortran class definition
- Defined in module
- Encapsulation (public, private)
- Element function/methods (keyword *contains*)
- Destructor (see C++, but not in Java)

oo.f90

```fortran
module oo_module
    implicit none
    private

    type, public :: oo
        private
        integer :: field_name
    contains
        procedure :: method_name
        final :: destructor
    end type oo

contains

    subroutine method_name(self)
        class(oo), intent(in) :: self
    end subroutine

    subroutine destructor(self)
        type(oo), intent(in) :: self
    end subroutine

end module oo_module
```

Fortran 2003: Inheritance using submodules

Example (from ISO/IEC TR 19767)

http://fortranwiki.org/fortran/show/Submodules

```
module points
  type :: point
     real :: x, y
  end type point

  interface
     module function point_dist(a, b) result(distance
       type(point), intent(in) :: a, b
       real :: distance
     end function point_dist
  end interface
end module points

submodule (points) points_a
contains
  module function point_dist(a, b) result(distance)
    type(point), intent(in) :: a, b
    real :: distance
    distance = sqrt((a%x - b%x)**2 + (a%y - b%y)**2)
  end function point_dist
end submodule points_a
```

Contents

Fortran is a compiled language

Workflow:

- Write Source code $\rightarrow$ *file.f*
- Compile $\rightarrow$ generates *file.o (machine-dependent)*
- Link with library if necessary $\rightarrow$ executable
- Run executable

This means:

- Compiler is needed: *gfortran* gnu Fortran compiler, or *ifort*,...

Different compiler options for optimization, checking for syntax, floating-point exceptions

Fortran Compiler

- GNU (in gcc - gnu compiler collection):
  - older versions: g77,g95
  - now gfortran
- NAG (Numerical Algorithms Group)
- Intel (ifort), IBM (Name: *xlf*) own Fortran compiler
- more: Lahey, Portland (RZ-Cluster)

Fortran compiler allow many things

ex1.f90

```fortran
program ex1
    real :: x(10)
    x = 0.0
    do i=1,11
        write(*,*) x(i)
    enddo
end program ex1
```

```
m-114:ArrayTest ts$ gfortran ex1.f90
m-114:ArrayTest ts$ ./a.out
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
   0.0000000
  3.70672815E-38
```

Example: no check of array bounds

Contents

Mixing languages

- Fortran subroutines and functions can be called from C and vice versa
- Compile source (e.g. with gcc and gfortran)and link object code with the same compiler
- Multi-dimensional arrays: Fortran stores column-wise, C row-wise
- Call-by-reference: always pass references/pointers from C to Fortran
- Java: JNI
- Python: F2py

Mixing languages: Some details may be important

do_something.f90

```fortran
integer function do_something(n)
   integer :: n
   do_something = 2*n
end function do_something
```

main.c

```c
#include <stdio.h>

extern int do_something_(int*);

int main() {
   int x=5,y;
   printf("vorher:  x = %i\n",x);
   y = do_something_(&x);
   printf("nachher: y = %i\n",y);
   return 0;
}
```

```
n-114:C_Fortran ts$ gfortran -c do_something.f90
n-114:C_Fortran ts$ gcc -c main.c
n-114:C_Fortran ts$ gcc main.o do_something.o -o a.out
n-114:C_Fortran ts$ ./a.out
vorher:  x = 5
nachher: y = 10
```

What is important?

- Fortran was the first higher programming language.
- It was designed for and is well-suited for high performance programming.
- It is used in many applications.
- Especially in climate models, Fortran is the most important language.
- Fortran standards have developed in time.
- Most important standards are F77 and F90 and newer.
- Modern Fortran is fully object-oriented.
- Advanced array operations.
- Many climate models are legacy codes.
- Mixtures exist.
- Free and commercial Fortran compiler exist.
- Might be insensitive to bad programming habits (in standard settings).