Karlsruher Institut für Technologie

Fortran 95/2003 Course

Robert Barthel / Hartmut Häfner
March 24, 2015

STEINBUCH CENTRE FOR COMPUTING - SCC

Development of Fortran Standards

Karlsruher Institut für Technologie

- The IBM Mathematical Formula Translating System
  - proposal by John W. Backus, 1953
  - first Fortran program Sept. 1954
  - first commercial compiler 1957 for IBM 704 mainframe
- 1956 FORTRAN
- 1958 FORTRAN II
- 1958 FORTRAN III
- 1962 FORTRAN IV
- 1966 FORTRAN 66
  - American Standard Fortran by American Standard Association, today ANSI
- 1978 FORTRAN 77
- 1992 Fortran 90
- 1997 Fortran 95
- 2004 Fortran 2003
  - actual Fortran 2003 ISO Standard
    - ISO/IEC 1539-1:2004(E)   340 CHF
- Fortran 2008 – Latest draft from 2010 (NO STANDARD)

http://en.wikipedia.org/wiki/Fortran

Open Source Fortran Compilers

Karlsruher Institut für Technologie

- gfortran
  - http://gcc.gnu.org/fortran/
  - Linux, Windows, Mac OS X
- g95
  - http://www.g95.org
    - Linux, Windows, Solaris, Mac OS X …
    - http://www.eclipse.org/photran/
    - includes already Coarray Fortran
- http://hpc.sourceforge.net/   for Mac OS X
- Open research for IA-64
  - http://ipf-orc.sourceforge.net/
  - http://www.open64.net/
- watcom
  - http://www.openwatcom.org/index.php/Main_Page
- Overview
  - http://www.fortran.com/metcalf.htm

Fortran Compilers (free for noncommercial use)

- ifort Compiler of Intel for non commercial use
  - https://software.intel.com/en-us/qualify-for-free-software
- Salford compiler (for Windows)
  - http://www.silverfrost.com/11/ftn95/overview.asp
- Sun compiler (?)
  - http://developers.sun.com/prodtech/cc/index.jsp

Important Commercial Fortran Compilers

Karlsruher Institut für Technologie

- Absoft Corporation
- Intel
- Lahey
- NAG (pure Fortran-Compiler)
- PathScale
- Portland Group
- Silverfrost FTN95

Links to Fortran Websites

Karlsruher Institut für Technologie

- http://www.fortran.de/
- http://de.wikipedia.org/wiki/FORTRAN
- http://www.fortran.com/
- Standardization committees WG5
  - international WG5: http://www.nag.co.uk/sc22wg5
  - US: http://www.j3-fortran.org
- Draft near to the 2003 Standard (May 10, 2004)
  - http://j3-fortran.org/doc/2003_Committee_Draft/04-007.pdf
- Draft near to the 2008/09 Standard
  - ftp://ftp.nag.co.uk/sc22wg5/N1701-N1750/N1723.pdf
- Intel compiler documentation (recommended)

  www.intel.com; search for ‚fortran language'

  https://software.intel.com/en-us/intel-parallel-studio-xe

Fortran References

Karlsruher Institut für Technologie

- Information Technology - Programming Languages – Fortran on-line or paper, the official standard, ISO/IEC 1539:1991 or ANSI x3.198-1992
- Fortran 90 Handbook, J. C. Adams et. al., McGraw-Hill, 1992 Complete ANSI/ISO Reference
- Fortran 95/2003 explained, M. Metcalf & J. Reid, Oxford University Press
- Fortran 90, Lehrbuch, D. Rabenstein, Carl Hanser Verlag
- Fortran 95, RRZN, Nachschlagewerk für ANSI-Fortran / DIN-Fortran / ISO/IEC 1539-1

Acknowledgments

Karlsruher Institut für Technologie

- Dr. Dave Ennis from Ohio Supercomputer Center (partially this course is based on)

- The Manchester and North HPC: Training and Education Centre
  - for material on which part of this course is based
  - Walt Brained, Unicomp, Inc. who contributed to this material and it is used with his permission

Fortran 90/95 Objectives

Karlsruher Institut für Technologie

- Language evolution
  - Obsolescent features
  - Keep up with "rival" languages: C, C++
- Standardize vendor extensions
  - Portability
- Modernize the language
  - Ease-of-use improvements through new features such as free source form and derived types
  - Space conservation of a program with dynamic memory allocation
  - Modularisation through defining collections called modules
  - Numerical portability through selected precision

Fortran 90/95 Objectives (2)

- Provide data parallel capability
  - Parallel array operations for better use of vector and parallel processors
  - High Performance Fortran (HPF) built on top of Fortran 90
- Compatibility with Fortran 77
  - Fortran 77 is a subset of Fortran 90
- Improve safety
  - Reduce risk of errors in standard code (improved argument checking)
- Standard conformance
  - Compiler must report non standard code and obsolescent features

Major new Features

Karlsruher Institut für Technologie

- Free-Form source style
  - Forget about those column numbers!
- User-defined Data Types
  - Heterogeneous data structure: composed of different "smaller" datatypes
- Array processing
  - Shorthand for loop nests and more....
- Dynamic memory allocation
  - Like C malloc & free
- Subprogram Improvements
  - Function/Subroutine Interface (like C Function Prototype)
  - Optional/Keyword Arguments

Major new Features (2)

Karlsruher Institut für Technologie

- Modules
  - Replace COMMON block approach to make "global" variables
  - Similar to C++ classes
- Generic Procedures
- Pointers
  - Long been needed

Other new Features

Karlsruher Institut für Technologie

- Specifications/ IMPLICIT NONE
- Parameterized data types (KIND)
  - Precision Portability
- Operator overloading/defining (like in C++)
- Recursive Functions
- New control structures
- New intrinsic functions (a fairly large library)
- New I/O features

Free Source Form

Karlsruher Institut für Technologie

- Type in any column you want!
- Line lengths up to 132 columns
- Lowercase letters permitted
- Names up to 31 characters (including underscore)
- Semicolon to separate multiple statements on one line
  - Don't confuse with C's use of ; to mark the end of a statement
- Comments may follow exclamation (!)

Free Source Form (2)

- Ampersand (&) is a continuation symbol
- Character set includes + < > ; ! ? % - " &
- New C-like relational operators: '<', '<=', '==', '/=', '>=', '>'

Free Source Form Example

Karlsruher Institut für Technologie

```
PROGRAM free_source_form
  ! Long names with underscores
  ! No special columns

  IMPLICIT NONE

  ! upper and lower case letters
  REAL :: tx, ty, tz

  ! trailing comment
  ! Multiple statements per line
  tx = 1.0; ty = 2.0
  tz = tx * ty

  ! Continuation symbol on line to be continued
  PRINT *, &
               tx, ty, tz

END PROGRAM free_source_form
```

Specifications

type [[, attribute]... ::] entity list

Type can be `INTEGER, REAL, COMPLEX, LOGICAL` or `CHARACTER` with optional kind value:

```
INTEGER ([KIND=] kind-value)
CHARACTER ([actual parameter list]) ([LEN=] len-value and/or [KIND=] kind-value)
TYPE (type name)
```

Attribute can be `PARAMETER, PUBLIC, PRIVATE, ALLOCATABLE, POINTER, TARGET, INTENT(inout), DIMENSION (`extent-list`), OPTIONAL, SAVE, EXTERNAL, INTRINSIC`

Specification Examples

- Can initialize variables in specifications

```
INTEGER :: ia, ib
INTEGER, PARAMETER :: rk=8, n=100, m=1000
REAL :: a = 2.61828, b = 3.14159
REAL(kind=rk) :: result_value
CHARACTER (LEN = 8) :: ch
INTEGER, DIMENSION(-3:5, 7) :: ia
COMPLEX(kind=rk), DIMENSION(n) :: z

result_value = 1.D0; z(1) = (1.5, 2.5)
```

Specification: CHARACTER

- Declaration:

```
CHARACTER [([LEN=]len, [KIND=]kind_param)] [, attr_list::]
         entity_list
```

- Examples:

```
CHARACTER*20                    name1     !* Fortran 77 style

CHARACTER                       name2*20  !* Fortran 77 style
CHARACTER(20)                   :: name3
CHARACTER(LEN = 20)             :: name4
CHARACTER(LEN = 5*len(name4))   :: name5

!* Static Arrays of Strings
CHARACTER(LEN = 20), dimension(100) :: name_list
```

Specification: CHARACTER (2)

- All character strings have fixed length

- Concatenation Operator //

```
CHARACTER(LEN=20) :: name
name= “Fred “// “die“
```

- Substring

```
CHARACTER(LEN=10) :: string
string = “Alphabet “
string(5:5) = “o“                    ! string = “Alphobet
string(4:)= “ine“                    ! string = “Alpine
PRINT * ,‘#‘, TRIM(string), ‘#‘      ! #Alpine#
```

- See intrinsic functions for string handling

`IMPLICIT NONE`

- In Fortran 77, implicit typing permitted use of undeclared variables.
  - This has been the cause of many programming errors.
- `IMPLICIT NONE` forces you to declare all variables.
  - As is naturally true in other languages
- `IMPLICIT NONE` may be preceded in a program unit only by `USE` and `FORMAT` statements (see order of statements).

`Kind` Values

Karlsruher Institut für Technologie

- 5 intrinsic types:
  `REAL, INTEGER, COMPLEX, CHARACTER, LOGICAL`
- Each type has an associated non negative integer value called the `KIND` type parameter
- Useful feature for writing portable code requiring specified precision
- A processor must support:
  - at least 2 kinds for `REAL` and `COMPLEX`
  - 1 kind for `INTEGER, LOGICAL` and `CHARACTER`
- Many intrinsics for inquiring about and setting kind values

`KIND` Value `REAL`

```
REAL (KIND = wp) :: ra    ! or
REAL (wp) :: ra
```

- Declare a real variable, `ra`, whose precision is determined by the value of the kind parameter, `wp`
- Kind values are system dependent
- An 8 byte (64 bit) real variable usually has kind value 8 or 2
- A 4 byte (32 bit) real variable usually has kind value 4 or 1
- Literal constants set with kind value: `const = 1.0_wp`

KIND Value REAL (2)

Karlsruher Institut für Technologie

- Common use is to replace DOUBLE PRECISION

```
INTEGER, PARAMETER :: idp = KIND(1.0D0)
REAL (KIND = idp) :: ra
```

- ra is declared as 'double precision', but this is system dependent
- To declare REAL in system independent way, specify kind value associated with precision and exponent range required:

```
INTEGER, PARAMETER :: i10=SELECTED_REAL_KIND(10, 200)
REAL (KIND = i10) :: a, b, c
```

- a, b and c have at least 10 decimal digits of precision and the exponent range 200 on any machine

`KIND` Values `INTEGER`

- Integers usually have 16, 32 or 64 bit
- 16 bit integer normally permits -32768 < i < 32767
- Kind values for each supported type
- To declare integer in system independent way, specify kind value associated with range of integers required:

```
INTEGER, PARAMETER :: i8=SELECTED_INT_KIND(8)
INTEGER (KIND = i8) :: ia, ib, ic
```

- `ia, ib` and `ic` can have values between 10^8 and -10^8 at least (if permitted by processor)

KIND Values INTRINSIC FUNCTIONS

Karlsruher Institut für Technologie

```
INTEGER, PARAMETER :: i8 = SELECTED_INT_KIND(8)
INTEGER (KIND = i8) :: ia
PRINT *, HUGE(ia), KIND(ia)
```

- This will print the largest integer available for this integer type (2147483674), and its kind value

```
INTEGER, PARAMETER :: i10 = SELECTED_REAL_KIND(10, 200)
REAL (KIND = i10) :: a
PRINT *, RANGE(a), PRECISION(a), KIND(a)
```

- This will print the exponent range, the decimal digits of precision and the kind value of `a`

Derived Types

Karlsruher Institut für Technologie

- Defined by user (also called structures)
- Can include different intrinsic types and other derived types
  - Like C structures and Pascal records
- Components accessed using percent operator (%)
- Only assignment operator (=) is defined for derived types
- Can (re)define operators - see later operator overloading
- Nested derived types
- Recursive type definition allowed if components are pointers

Examples of Derived Types

Karlsruher Institut für Technologie

```
type person
   integer         :: age
   character (57) :: name
end type person

integer, parameter  :: groupsize=50

type group
   type (person) :: employee(groupsize)
   real          :: salary(groupsize)
   integer       :: number_of_employees
end type group
```

```
type(person)                     :: person_a
type(person), dimension(1:3) :: persons
type(group)                      :: working_unit
```

Examples of Derived Types (2)

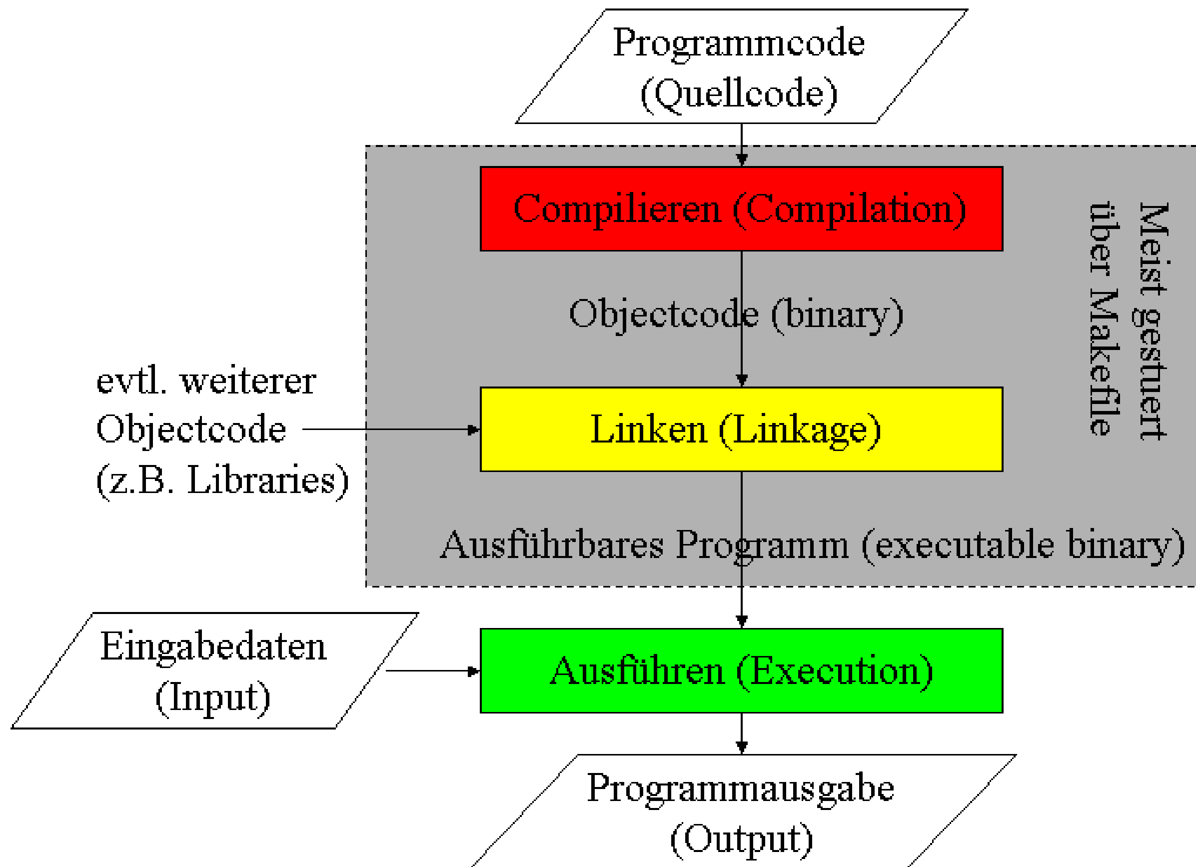
```
Person_a%age=35; person_a%name='Peter Smart'

! structure constructor is not recommended,
! use keyword arguments instead
Persons(2) = person(27,'Hugo Schmitt')

! Fortran 2003
persons(2)=person(age=27,name='Hugo Schmitt')

working_unit%employee(44)%age = 55
```

Compilation - Linking



Compilation and Linking with Intel-Compiler

- We are using the Intel-Compiler 14.0.4 on bwUniCluster
  - Compiler loaded by command
    `module load compiler/intel`
    (`module list` – shows all loaded modules)

- Compilation and linking by the command
  ifort -o <progname> <progname>.f90

Fortran Suffix Names

- Suffices for the Fortran-Compiler from Intel

Command	File name suffix	Source format	Language level
`ifort`	`.f .ftn .for .i`	`-fixed -72`	`-nostand`
`ifort`	`.F .FTN .FOR .fpp .FPP`	`-fixed -72 -fpp`	`-nostand`
`ifort`	`.f90 .i90`	`-free`	`-nostand`
`ifort`	`.F90`	`-free -fpp`	`-nostand`

Compiler Options (for the Intel-Compiler)

Karlsruher Institut für Technologie

-c	compile only, do not link the object codes to create an executable program.
-Ipath	specify a directory, which is used to search for module files and include files, and add it to the include path.
-g	include information for a symbolic debugger in the object code.
-O [level]	create optimized source code. The optimization levels are 0, 1, 2, and 3. The option -O is identical to -O2. Increasing the optimization level will result in longer compile time, but will increase the performance of the code. In most cases at least optimization level -O2 should be selected. The -O2 option of the Intel compilers enables optimizations for speed, including global code scheduling, software pipelining, predication, and speculation.
-p or -pg	create code for profiling with the gprof utility.
-Lpath	tell the linker to search for libraries in path before searching the standard directories
-llibrary	use the specified library to satisfy unresolved external references
-o	name specify the name of the resulting executable program.

Separate Compilation and Linking

Karlsruher Institut für Technologie

- Compilation

```
ifort -c <prog1>.f90  <prog2>.f90  …

ifort -c -I<directory_with_modules> <prog1>.f90
<prog2>.f90  ...
```

- Linking

```
ifort -o <exe_name> <prog1>.o <prog2>.o ...
```

Program Development