Syllabus for 5 Day Fortran 90 Course

AC Marshall

September 2, 1997

Abstract

The course is scheduled for 5 days. The timetable allows for two sessions a day each with a one hour lecture and a two hour practical session. The first session is intended to include local infomation about the operating system, editors and Fortran 90 compilers.

The following topics will be covered: data types and declarations, expressions, control constructs, I/O, array syntax, intrinsics, program units, `MODULE`*s, pointers, derived types, object-oriented programming, parameterised data types.*

The package contains lecture slides, teachers notes annotating the corresponding overhead slide, course notes, exercise sheets (one per practical session) and solutions (including compilable programs).

1 Introduction

FORTRAN 77 was always the first choice for scientific and engineering software due, in the main, to two reasons:

- compiler availability;
- efficiency

Fortran 90 was designed with the latter point very much in mind (which explains some of the restrictions imposed in the standard). Since its definition and standardisation, it became clear that the functionality of FORTRAN 77 needed to be extended in order to keep in line with current software engineering practises and it was out of this need that Fortran 90 emerged.

2 Course Format

The course is comprised of the following:

- instructions for use,
- this syllabus,
- OHP originals (for photocopying onto slides),
- lecturers guide (notes for guidance),
- student notes,
- exercises,
- exercises with solutions,
- compilable solution programs,
- resource list.

The lecture course will take place between 9am and 5pm each day of the week. The basic format of the course will be the same each day:

Time	**Event**
9:00–10:00	morning lecture
10:00–10:30	morning break
10:30–12:30	morning practical session
12:30–1:30	lunch break
1:30–2:00	afternoon lecture
2:00–2:30	afternoon break
2:30–5:00	afternoon practical session

When a lecture is longer or shorter than one hour, the corresponding breaks and practical sessions should be shifted backwards or forwards correspondingly.

3 Syllabus

The syllabus and lecture content is given below.

Session 1: Introduction

Topics covered:

- ☐ language evolution
  - ◇ standardisation process.
- ☐ drawbacks of FORTRAN 77
  - ◇ fixed source form, absence of parallelism, lack of dynamic storage, non-portability, lack of user-defined data type, lack of explicit recursion, reliance on unsafe storage association features.
- ☐ Fortran 90 new features
  - ◇ free source form, array syntax, array intrinsics, dynamic storage and pointers, portable data types, derived data types and operators, recursion, object based programming (`MODULE`s), procedure interfaces, new control structures, user defined generic procedures, enhanced I/O.
- ☐ advantages of new Fortran 90 features
  - ◇ how unsafe FORTRAN 77 features map to new Fortran 90 syntax.
- ☐ obsolescent features of Fortran 90
  - ◇ philosophy and Fortran 90 equivalents, list of obsolescent features.
- ☐ undesirable features
  - ◇ things which are not yet obsolete but which should be avoided: fixed source form layout, implicit declaration of variables, `COMMON` blocks, assumed size arrays, storage and sequence association, `ENTRY` statements and the computed `GOTO` statement.
- ☐ object oriented facilities
  - ◇ brief explanation of: data abstraction, data hiding, polymorphism, reusability, encapsulation, inheritance and extensibility.

Session 2: Elements of Fortran 90

Topics covered:

- ☐ example of a Fortran 90 program
  - ◇ a line by line dissection of a Fortran 90 program.
- ☐ coding style
- ☐ Fortran 90 source form
- ☐ character set
- ☐ significant blanks
  - ◇ free form source code rules, examples, mandatory and optional blanks.
- ☐ comments
  - ◇ the `!` initiator, free source form comments, `!`s in strings, formatted comments, for example, HPF systems.
- ☐ valid Fortran object names
- ☐ statement ordering
- ☐ intrinsic types
  - ◇ object classes: character, boolean, numeric; intrinsic types.
- ☐ literal constants
  - ◇ examples of data types and names.
- ☐ implicit typing
  - ◇ dangers, `IMPLICIT NONE` statement.
- ☐ numeric and logical declarations
  - ◇ explicit typing, syntax, attribute lists and examples.
- ☐ character declaration
  - ◇ single character objects, strings, `LEN=` specifier, overriding the `LEN=` specifier using `*` notation.
- ☐ symbolic constants (`PARAMETER`s)
  - ◇ attributed declaration, `CHARACTER` constants can inherit their length, benefits and examples.
- ☐ scalar initialisation
  - ◇ initialisation within declarations, padding (for `CHARACTER` s), `PARAMETER`s, permitted intrinsics in initialisations, examples.
- ☐ scalar expressions
  - ◇ intrinsic operators, use of expressions and examples.
- ☐ scalar assignment
  - ◇ the assignment operator, examples.
- ☐ intrinsic numeric operations
  - ◇ mathematical operators: `**`, `*` and `/`, `+` and `-` (monadic and dyadic). Relational operators: `.GT.`, `.GE.`, `.LE.`, `.LT.`, `.NE.`, `.EQ.` and the equivalent: $>$, $>=$, $<=$, $<$, $/=$, $=$. Examples.

- ☐ intrinsic logical operations
  - ◇ logical expressions, operators: `.NOT.`, `.AND.`, `.OR.`, `.EQV.` and `.NEQV.`, examples.

- ☐ character operations
  - ◇ substrings: specification of part of a string, differences to array sections. Concatenation: the // operator.

- ☐ operator precedence
  - ◇ implications for expressions, pitfalls, relative precedence and examples.

- ☐ precision errors
  - ◇ how real numbers are stored, round-off errors, catastrophic accuracy loss and examples.

Session 3: Control Constructs, Intrinsics and Basic I/O

Topics covered:

- control flow
  - conditional execution statements and constructs, loops and multi-way choice construct.
- `IF` statement
  - syntax, execution process, restrictions, examples of use and visualisation.
- `IF` construct
  - syntax, execution process, restrictions, nesting, examples of use and visualisation.
- nested and named `IF` constructs
- conditional exit loops
  - `EXIT` clause, restrictions.
- `CYCLE` statement
- named and nested loops
  - loops name syntax, named `EXIT` and `CYCLE` statements, scope and example
- `DO` ... `WHILE` loops
  - semantics, redundancy, restrictions.
- indexed loops
  - `DO` loops, syntax, number of iterations, execution process, missing stride, examples, scope of index variables, final values of index variables.
- the `SELECT CASE` construct
  - comparison to `IF` construct, example, visualisation.
- mixed type numeric expressions
  - promotion, examples.
- mixed type assignment
  - truncation, coercion, examples.
- integer division
  - mixed type arithmetic, dangers, short cuts, truncation.
- Intrinsic Procedures
  - classes, functions, subroutines.
- type conversion functions
  - `REAL`, `INT`, `DBLE`, `IACHAR` and `ACHAR`.
- mathematical intrinsic functions
  - `ASIN`, `ACOS`, `ATAN`, `ATAN2`, `TAN`, `COS`, `SIN`, `TANH`, `COSH`, `SINH`, `EXP`, `LOG`, `LOG10`, `SQRT`.
- numeric intrinsic functions
  - `ABS`, `AINT`, `ANINT`, `CEILING`, `FLOOR`, `CMPLX`, `DBLE`, `DIM`, `INT`, `MAX`, `MIN`, `MOD`, `MODULO`, `REAL` and `SIGN`.

- character intrinsic functions
  - ACHAR, ADJUSTL, ADJUSTR, CHAR, IACHAR, ICHAR, INDEX, LEN, LEN_TRIM, LGE, .., LLT, REPEAT, TRIM and VERIFY.
- scalar I/O
  - PRINT statement, READ statement, I/O of numeric, logical and character objects, examples.

Session 4: Arrays

Topics covered:

- ☐ basic array terminology
  - ◇ visualisation, rank, bounds, extent, size, shape and conformance, declarations, `DIMENSION` attribute, default lower bounds, zero-sized arrays, indexing.
- ☐ array conformance
  - ◇ scalar conformance, conformance between arrays, difference between size and shape.
- ☐ array element ordering
  - ◇ FORTRAN 77 storage association, conceptual column major form.
- ☐ array syntax
  - ◇ whole arrays, elements, sections.
- ☐ whole array expressions
  - ◇ operations between conformable arrays, the concept of parallel execution, elemental intrinsic function invocation, examples.
- ☐ array sections
  - ◇ specification and visualisation.
- ☐ array sections
  - ◇ subscript-triplets, default subscript-triplet values, examples of triplets: strided sections, zero-sized sections, reversed sections.
- ☐ array I/O
  - ◇ array element ordering, example.
- ☐ array inquiry intrinsics
  - ◇ inquiry intrinsics in procedures, `LBOUND` and `UBOUND`, `SHAPE`, `SIZE` and `ALLOCATED`.
- ☐ array constructors
  - ◇ `(/ .. /)` constructors, implied `DO` statements, `RESHAPE`.
- ☐ `RESHAPE` intrinsic function
- ☐ initialising arrays
  - ◇ constructors in declarations, `RESHAPE` in declarations.
- ☐ deferred-shape arrays
  - ◇ heap storage, `ALLOCATABLE` attribute, `ALLOCATE` statement, `STAT=` specifier, reclaiming storage, `DEALLOCATE` statement, `ALLOCATED` function, memory leakage
- ☐ masked assignment
  - ◇ `WHERE` statement, masks, `WHERE` construct, execution process, example.
- ☐ vector-valued subscripts
  - ◇ restrictions, examples.
- ☐ random number intrinsic

- vector and matrix multiply intrinsics
  - DOT_PRODUCT and MATMUL. Examples of use.

- maximum and minimum intrinsics
  - MAX and MIN intrinsics, examples.

- array location intrinsics
  - MAXLOC and MINLOC intrinsics, examples.

- array reduction intrinsics
  - the concept of reduction functions, ALL, ANY, COUNT, MAXVAL, MINVAL, PRODUCT and SUM, the optional DIM argument.

Session 5: Procedures

Topics covered:

- program units
  - PROGRAMs, MODULEs, SUBROUTINEs and FUNCTIONs.

- main program
  - syntax, visualisation of scope, example.

- procedures
  - intrinsics, libraries of procedures.

- internal subroutines and functions
  - examples, syntax, invocation, argument association, local objects.

- argument intent
  - INTENT attribute, meaning of INTENT(IN), INTENT(OUT) and INTENT(INOUT). Benefits of using intent, example of use.

- scope
  - what scope means, host- and use-association.

- host association
  - scope of local variables in a procedures, variables with the same name at different scoping levels, dummy arguments and host association.

- static storage class
  - SAVE statement / attribute, local objects.

- keyword arguments
  - argument association: (positional / keyword), keyword names, restrictions, examples and benefits.

- optional arguments
  - philosophy, syntax, restrictions and example.

Session 6: More Procedures

Topics covered:

- ☐ dummy array arguments
  - ◇ explicit-shape arrays, assumed-shape arrays.
- ☐ explicit-shape arrays
  - ◇ syntax, restrictions, inflexibility.
- ☐ assumed-shape arrays
  - ◇ recommended method, example, restrictions.
- ☐ automatic arrays
  - ◇ temporary local objects, declaration, restrictions.
- ☐ `SAVE` statement and allocatable arrays
- ☐ explicit-length character dummy arguments
  - ◇ syntax, restrictions, inflexibility.
- ☐ assumed length character dummy arguments
  - ◇ inheritance of object length from actual argument, assumed-length character dummy argument, examples.
- ☐ array-valued functions
  - ◇ specifying result variable, inheriting size of result from arguments, declaration of function (explicit interface).
- ☐ character-valued functions
  - ◇ declaration of function result, inheriting size of result from arguments, example.
- ☐ side effect functions
  - ◇ non-determinacy of side effect functions, execution order, examples.
- ☐ recursive procedures
  - ◇ the concept of recursion, benefits, drawbacks, recursive subroutines, recursive functions (`RESULT` keyword), syntax, practical example.
- ☐ software resusability - modules
- ☐ data hiding
  - ◇ objects with `PRIVATE` and `PUBLIC` attributes
- ☐ `USE` statement
  - ◇ use-association, name clashes, use only clause, examples.
- ☐ modules — visualisation
- ☐ overview of modules
  - ◇ global object declaration, interface declaration, procedure declaration, controlled object accessibility, packages of whole sets of facilities and semantic extension.

Session 7: Pointers and Derived Types

Topics covered:

- introduction to pointers
  - the concept of pointers and targets, pointer states, `ASSOCIATED` intrinsic and visualisation of pointers in memory.
- pointer status
  - three different states: associated, disassociated, undefined.
- pointer declaration
  - `POINTER` attribute, strong typing of pointers.
- targets of pointers
  - `TARGET` attribute, declaration of targets, examples.
- pointer manipulation
  - how to use pointers, => and = operators.
- pointer assignment
  - the difference between => and =, restrictions and examples.
- pointers to arrays
  - strong typing, association with regular sections, restrictions, visualisations and examples.
- dynamic allocation of pointer targets
  - implications, restrictions.
- automatic attributing of pointers
  - pointers have the `TARGET` attribute.
- pointer association
  - `ASSOCIATED` intrinsic, results of function call, problems of undefined pointers, association with a specific target.
- pointer disassociation
  - `NULLIFY` and `DEALLOCATE`.
- pointers to arrays and allocatable arrays
  - subtle differences between pointers to arrays and allocatable arrays, efficiency considerations, difference in implementation.
- pointer example
  - practical example of where pointers increase efficiency.
- pointer arguments
  - the two different interpretations depending on dummy argument declaration, mandatory interface, restrictions and example.
- pointer valued functions
- pointer I/O

- derived types
  - philosophy, definition, supertypes.

- assignment to derived type objects
  - assignment between objects of same defined type, assignment between objects of a different defined type: component by component assignment using the % operator, whole object assignment using a constructor.

- arrays and derived types
  - derived types containing non-`ALLOCATABLE` arrays, arrays of derived type objects, restrictions on reference.

- derived type I/O
  - output format, restrictions.

- derived types and procedures
  - definition, host / use association.

- derived type valued functions

Session 8: Modules and Object-based Programming

Topics covered:

- □ pointer components of user-defined types
- □ pointers and recursive data structures
  - ◇ linked lists, assignment between structures containing pointer components.
- □ practical example of linked lists
- □ arrays of pointers
  - ◇ how to declare arrays of pointers, restrictions on use, visualisation.
- □ modules — object-based programming
  - ◇ introduction to program packaging and encapsulation.
- □ derived type constructors in modules
- □ user-defined generic interfaces
  - ◇ generic interface syntax, encapsulation — module procedures, restrictions, overload sets, resolution of invocation, example.
- □ derived type output procedures
- □ extending generic intrinsic interfaces
- □ operator overloading
  - ◇ syntax, monadic and dyadic overloads and restrictions.
- □ example of operator overloading
- □ defining new operators
  - ◇ operator syntax, monadic and dyadic defined operators, syntax considerations and restrictions.
- □ example of defining new operators
- □ user defined operator precedence
- □ user-defined assignment
  - ◇ explicitly programmed user-defined assignment, overloading the = operator, assignment overload sets, syntax restrictions, redefinition of previous overloads.
- □ example of user-defined assignment
- □ semantic extension
  - ◇ combining: derived types, visibility, overloading.
- □ semantic extension

Session 9: Parametrised Intrinsic Types Plus

Topics covered:

- ☐ complex data type
  - ◇ complex objects, declarations, symbolic constants, constants, CMPLX intrinsic, complex expressions.
- ☐ complex intrinsic functions
  - ◇ AIMAG, REAL or DBLE, CONJG, ABS, SIN etc.
- ☐ parameterised data types
  - ◇ kind selection, numeric portability.
- ☐ integer kind selection
  - ◇ SELECTED_INT_KIND intrinsic function, specification of numeric range, default kinds, portability.
- ☐ constants of a selected integer kind
  - ◇ syntax, portability.
- ☐ real kind selection
  - ◇ SELECTED_REAL_KIND intrinsic, selecting precision and exponent range, portability, COMPLEX data types, DOUBLE PRECISION data types (use REAL).
- ☐ KIND function
- ☐ mixed kind expression evaluation
  - ◇ kind promotion, pitfalls, examples.
- ☐ logical kind selection
  - ◇ selecting different logical kinds, implications.
- ☐ characters of different kinds
  - ◇ default character set, selection of different character sets, restrictions, mixing character sets, specification of literals, intrinsic functions.
- ☐ kinds and procedure arguments
  - ◇ arguments correspondence, generic interfaces.
- ☐ bit variables and bit manipulation intrinsic functions
  - ◇ representation of bit variables. Intrinsics: BTEST, IAND, IBCLR, IBITS, IBSET, IEOR, IOR, ISHFT, ISHFTC, NOT and MVBITS. Examples of use.
- ☐ array construction intrinsics
  - ◇ MERGE, SPREAD, PACK and UNPACK.
- ☐ MERGE intrinsic
- ☐ SPREAD intrinsic function
- ☐ PACK intrinsic
- ☐ UNPACK intrinsic
- ☐ TRANSFER intrinsic

Session 10: Odds and Ends

Topics covered:

- input / output
  - logical units, I/O streams, properties of streams: filename, action, status, and access method.
- the `OPEN` statement
  - the following specifiers: `UNIT=`, `FILE=`, `ERR=`, `STATUS=`, `ACCESS=`, `ACTION=`, `RECL=`.
- `READ` statement
  - Non-advancing I/O, the following specifiers: `FMT=`, `IOSTAT=`, `ERR=`, `END=`, `EOR=`, `ADVANCE=`, `REC=`, `SIZE=`. Examples of use.
- `WRITE` statement
  - non advancing input, the following specifiers: `UNIT=`, `FMT=`, `IOSTAT=`, `ERR=`, `ADVANCE=`, `REC=`.
- `FORMAT` statement and `FMT=` specifier
- edit descriptors
  - `I, F, E, L, A` and `X` descriptors, example of use, overflow.
- miscellaneous I/O statements
  - overview of: `CLOSE`, `REWIND`, `BACKSPACE` and `ENDFILE`.
- external procedures
  - separate compilation, requirement of explicit interfaces, use as procedure arguments.
- subroutines
  - syntax, internal subroutines, invocation, declaration, arguments (actual and dummy), examples.
- functions
  - syntax, examples of declaration and use.
- procedure interfaces
  - `INTERFACE` blocks, syntax, contents, internal procedure interfaces, examples, benefits.
- required interfaces
  - instances where is procedure interface is mandatory, benefits of always using an interface.
- procedure arguments
  - declaration of procedures, `INTRINSIC` statement, `EXTERNAL` statement, syntax of procedure declarations, explicit typing of functions, restrictions and examples.
- `DATA` statement
  - syntax, use, automatic attributing, example.
- data statement and the implied `DO` loop
- `GOTO` statement
  - dangers of use, logical spaghetti, unreadability, power, examples.
- `RETURN` and `STOP` statements

- Fortran 95
  - FORALL, nested WHERE, ELEMENTAL and PURE procedures, user-defined functions in initialisation expressions, automatic deallocation, deleted features, obsolescent features, language tidy-ups and ambiguities. Rationale for changes.

- overview of High Performance Fortran
  - distributed memory processing, philosophy. Syntax extensions: FORALL, new intrinsics, PURE and ELEMENTAL procedures, compiler directives: PROCESSORS, ALIGN, TEMPLATE, DISTRIBUTE, distribution methods, and examples.