INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/841-6

FOURTH ICTP-URSI-ITU(BDT) COLLEGE ON RADIOPROPAGATION: Propagation, Informatics and Radiocommunication System Planning

30 January - 3 March 1995

Miramare - Trieste, Italy

Introduction to Fortran

A. Nobile
ICTP - Trieste, Italy

Introduction to Fortran
FORTRAN 77

Structure of the language

A FORTRAN program is composed of a main program and one or more subprograms

```
  PROGRAM TEST
.....
  END
  SUBROUTINE M1(X,Y)
.....
  END
  FUNCTION F(Z,T)
....
  END
```

- Compiled language
- The main program is executed by the system; it can 'call' its subprograms (that can be SUBROUTINES or FUNCTIONS).
- One and only one main program!
- One or more files (program units can be rearranged in multiple files freely!)

✦ **independent compilation**

A scheme of the structure of a FORTRAN program

Input some values *(sometimes omitted)*
Compute some new values
Output the new values

Values of which type?
How we 'compute new values'?
How do we input and output values?

The FORTRAN data types

INTEGERS
REAL
DOUBLE PRECISION
COMPLEX
(DOUBLE COMPLEX)
LOGICAL
CHARACTER

INTEGER:
Integer values between -N and +M (N,M implementation dependent: on most machines, $N=2^{31}-2*10^9$ and $M=2^{31}-1$). Obey standard integer arithmetic unless overflow.

REAL
The standard floating point arithmetic of the machine. Now, on essentially al the machines, conformant to the IEEE standard. Rational numbers of the form

$+/- \ 2^{m}*p$

where

$-127<m<128,$

and p is a **binary** fraction of the form
$0.1p_1p_2...p_{23}$ (p's 0 or 1)

- Not *real*: a subset of rationals
- Relative accuracy: difference between one and the next about $2^{-24} \sim 10^{-7}$ of its value
- Range about $2^{-127}<x<2^{127} \sim 10^{-38}<x<10^{38}$, plus the 0.0
- Inexact arithmetic
- Non-associative arithmetic

DOUBLE PRECISION
Like the REAL, but with extended accuracy (usually about $2^{-52} \sim 10^{-16}$) and range ($2^{-1024}<x<2^{1024} \sim 10^{-308}<x<10^{308}$)

COMPLEX
A couple of REAL, manipulated according to the arithmetic of complex numbers

DOUBLE COMPLEX
An NON-STANDARD extension (in FORTRAN 77): A COMPLEX built using DOUBLE PRECISION instead of REAL

LOGICAL
An type that contains only two values (true and false). Manipulated with the operators and the rules of Boolean algebra.

CHARACTER
A string of (ASCII) characters, of fixed length. The only operation admitted is chaining two such strings together. There are functions allowing for the extraction of a substring or the search of characters in the string.

Constants and variables

A FORTRAN object can be a CONSTANT or a VARIABLE.

A CONSTANT can be:

1) a literal constant

```
  A=1.0
C 1.0 (like 1.0e0 or 1. or 1.e0)
C is a literal real constant
  N=3
C 3 is a literal INTEGER constant
  X=3.5d0
C 3.5d0 is a literal DOUBLE PRECISION
C constant
  LOGVAL=.TRUE.
C .TRUE. and .FALSE. are THE logical
C constants
  CVAL=(0.0,1.0)
C (0.0,1.0) is a literal COMPLEX
C constant
  PRINT *, 'THIS IS A STRING'
C literal character constant: string
C in single quotes
```


2) or a NAMED constant (called also a PARAMETER)

```
  PROGRAM ALFA
  PARAMETER (N=20, PI=3.14125)
C  N and PI are named constants
....
  IF (I .LT. N) THEN
.....
C  means if I LessThan N
   X=PI
...
  ENDIF
...
  END
```

Variables

A **variable** is something that has a type, and can hold a value of that type. The value can be stored into the variable by an **assignment**

```
A=3.5
```

or by an **INPUT** operation

```
READ *, X
```

Before the first such operation, the value of a variable is ***undefined***

The names in FORTRAN

A name is composed of any sequence of letter or digits, must start with a letter and must be at most 6 characters long (crazy standard: most compilers accept 31 characters...)
A variable cannot have the same name of a named constant.
The names of variables and named constants are known only within the *program unit where they appear.*

Types and declarations

A variable (and a named constant) can (should) be declared. Declarations have the form

```
REAL A,K,KMAX
INTEGER L,AN
```

They must be at the beginning of the program, before any *executable statement*

If a variable is not declared, its type is taken from the first letter of its name:

- if the first letter is between I and N, it is assumed to be an INTEGER
- else it is assumed to be a REAL

BUT this rule can be changed by an IMPLICIT statement


```
  IMPLICIT REAL (K-L)
  IMPLICIT COMPLEX C
...
  C1=(1.0,0.0)
  CI=(0.0,1.0)
  K1=0.3
  I=45
```

IMPLICIT statements must be placed at the beginning, before all the declarations, but after the PROGRAM, FUNCTION or SUBROUTINE statements

Good practices (but not standard):
use IMPLICIT NONE (A-Z)
to defeat automatic typing and declare everything (protects against misspelling!)

Other things with a value

Function invocations

Two type of functions:

1) User-defined (see above and later)
2) INTRINSIC FUNCTIONS

```
....
DOUBLE PRECISION Y
REAL X
COMPLEX C
Y=SIN(1.0d0)
X=SIN(1.0E0)
C=SIN((1.0,0.0))
......
```

GENERIC: Take argument of any (reasonable) type and return value of same type. Correct operations selected by the compiler on the basis of the type of the argument!

Most important intrinsic functions:
SQRT SIN COS TAN ACOS ASIN ATAN EXP LOG SINH COSH TANH ASINH ACOSH ATANH MAX MIN MOD ABS
Note: MAX,MIN accept variable number of arguments. But MAX(1.0d0, 3) impossible...

Other intrinsic functions: type conversion functions:
take argument of any type and convert to the type of destination:
INT REAL DBLE CMPLX DCMPLX IMAG
Note:

```
C=(0.,1.)
C=CMPLX(R1,R2)
```

Ambiguity about DOUBLE COMPLEX, because not standard. Compiler dependent.

Expressions

What to do with values? Compute expressions!

Arithmetic operators:

** (exponentiation)
* /
+ - (unary or binary)

"Reasonable" behaviour:
precedence is the 'usual' one: first **, then * and /, then + -
Associativity is left-to-right (except **)
Various types can be mixed, and are converted to 'most powerful' type before expression evaluation.

```
      DOUBLE COMPLEX X, C, T
      DOUBLE PRECISION Z
      Z=0.34d0
      N=6
      X=Z*(0.5,1.4)+C/N
C DOUBLE * COMPLEX: both converted to
C DOUBLE COMPLEX
C COMPLEX/INTEGER: both converted to
C COMPLEX before division
C COMPLEX+DOUBLE COMPLEX: both
C converted to DOUBLE COMPLEX before
C sum
      T=CMPLX(1.0d0,Z)+X/2
C why here CMPLX and before not?
C WARNING
      X=1/4+1/4+1/4+1/4
```


Assignment

variable=expression
"Reasonable": *expression* is evaluated, then, if needed, it is converted to the type of *variable*, and is stored in *variable*

Warning:
X=3/5+4/5
gives 0.0!

Other expressions

Relational operators

.LT. .LE. .EQ. .NE. .GE. .GT.

Take arithmetic operands and return a LOGICAL value (.TRUE., .FALSE.)

```
LOGICAL CONVERGENCE
....
CONVERGENCE = XOLD-XNEW .LT. 1.e-4
....
```

Logical operators
Take one or two logical operands and return a logical value

.NOT.
.AND.
.OR.
.XOR. .EQV. .NEQV.

Basic Input-Output

A value (or a set of values) can be obtained by READING an external *file (device)*and can be used to be WRITTEN to an external *file(device)*

To the FORTRAN program, all external sources/destinations of data appear as
LOGICAL UNITS
identified by a (usually small) integer number

```
READ(10)X,Y,Z
WRITE(20)N
```

Therefore:
- I-O operations to associate a logical unit to a file/device
- I-O operations on logical units

✦ **Abstraction:**
details of devices/files etc. hidden from the program. OS would not be enough because:

✦ **portability:**
Operating system dependencies hidden from the program !

Formatted I-O

Simple case: value has to be read/was written by a human:

EXTERNAL FORM: string of text
INTERNAL FORM: whatever used by the hardware
=> Format conversion!

The standard units

In most systems, human interaction assumed as:

keyboard input console output

possibly redirected (DOS:

```
a.exe <input.dat >output.dat
```

)

Therefore: two standard channels, one for formatted input, one for formatted output, always available.

In FORTRAN these units:

- DO NOT REQUIRE ANY COMMAND TO BE CONNECTED (PRE-CONNECTED)
- ACCORDING TO THE STANDARD, ARE INDICATED BY '*' IN THE **READ and WRITE** STATEMENTS
- IN MOST SYSTEMS, ALSO AVAILABLE AS **5** (standard input) and **6**(standard output)

```
READ (*,*) X
WRITE(*,*) X,X**2
```

or

```
READ(5,*)X
WRITE(6,*)X,X**2
```

Structure of READ and WRITE:
READ(*unit, format*)*list_of_variables*
WRITE(*unit, format*)*list_of_expressions*

The simplest format: the list-directed I-O

An asterisk as a format specifier (see above): you do not specify how the conversion to/from text should be performed: the system uses a 'reasonable default', depending from the *list* of items to be read or written.

On input: items are separated by sequences of blanks. Integers are just sequences of digits, reals are like REAL constants, DOUBLE are like REAL or DOUBLE constants...

On output: items are converted to some 'reasonable' form and issued, separated by a blank and terminated by a newline.

Recommendation:
To be used practically always on input.

On output, could be impractical only for tables or other large amounts of output, where regular layout is useful.

Variants
READ *,*list*
instead of
READ(*,*)*list*

PRINT *,*list*
instead of
WRITE(*,*)*list*

Other units: the OPEN statement

Associates a LOGICAL UNIT (known to FORTRAN) to a FILE (known to the Operating System).
Specifies aspects of the behaviour of the unit

```
      OPEN(8, FILE='MYS.DAT',
     $    FORM='FORMATTED',
     $    ACCESS='SEQUENTIAL' ,
     $    STATUS='OLD')
.....
      CLOSE(8)
```

FILE:
  CHARACTER expression (constant or variable: could have been read before!)
STATUS:
  'OLD'
  'NEW'
  'UNKNOWN'
ACCESS:
  'SEQUENTIAL'
  'DIRECT'
FORM:
  'FORMATTED'
  'UNFORMATTED'

Binary I-O

If data have to be read/were written by another program, **on the same type of machine**:

avoid FORMAT CONVERSION
overhead(large!) ; creates smaller files.

SPECIFIED BY OMITTING FORMAT SPECIFICATION:

```
READ(9)X,Z
WRITE(11)P
```

If unit was opened with OPEN, that OPEN had to include FORM='UNFORMATTED'

Default file-to-unit association

Compiler dependent. On this one, uses the arguments of the execution command, in the order they are met (Dangerous... Use explicit OPEN)

The form of the statements

"Initial statements" (normal statements)

label		text	comment
1 5	6	7 72	

"Continuation" statements

	X	text	
1 5	6	7 72	

Comments

C *	comment text
1	

Remarks

On most compilers :

- Comment after column 73 cannot be used any longer on most machines. Leave blank.
- Instead of initial blanks
  label \tab *statement*
  or
  \tab *statement*
- option to extend input line beyond column 72 (not on this compiler)

Blanks not significant!


```
  A    e=1
C same as
  AE=1
C
  DO 1 I=1 100
C means
  DO1I=1100
C
  IF(...)THEN
    ....
  ELSE I=2
    .....
  ENDIF
C causes ELSE to disappear (ELSEI=2)
```



```
C Sample of program
  program test
  complex j
  parameter (j=(0.0.1.0))
  real r, l, c, omega
  complex z
  write(*,*) 'Enter r, l, c, omega'
  read (*,*) r,l,c,omega
  Z=r+j*(omega*l -1/(omega*c))
  write(*,*)'abs(z)=',abs(z)
  end
```

The compiler (here)

```
FL /FPi87 /G2 prog.for
prog.exe
```

/FPi87 use hardware floating point
/G2 use post-80286 instruction set

Other relevant options

/Ox full optimization
/4Yb turn on debugging

Part II: The control and data structures
Statements executed sequentially.

- Conditional
- Loop
- GOTO

The control structures

GOTO

GOTO *label*

label number (1-5 digits)

label *statement*

```
......
  GOTO 45
.....
45 CONTINUE
```

Labels are known only within the program unit where they appear!

CONDITIONAL

IF(*logical expression*)THEN
.....
ELSE IF(*logical expression*) THEN
.....
ELSE
.....
ENDIF

```
READ (*,*)Z
IF(Z.GT.0)THEN
 WRITE(*,*)SQRT(Z)
ELSE
 WRITE(*,*)'ILLEGAL VALUE'
ENDIF
```

- Jumping inside an IF block is forbidden!

Variant

IF(*logical expression*)*statement*

```
IF(Z.EQ.0.0)STOP
IF(Z.LT.0.0) GOTO 100
```


LOOP
DO ***label ctrl=in, fin, incr***
.....whatever
label **CONTINUE**

ctrl : INTEGER variable
in, fin, incr: integer values. If incr omitted, 1 assumed

MEANS (EXACTLY!)

```
    ctrl=in        ! initialize ctrl variable
    N.incr=incr    ! store increment
    N.iterations=(fin+incr -1-in)/incr
                   ! compute number of iterations
    N.done=0       ! set number of completed
                   ! iterations to 0
1   IF(N.done .LT. N.iterations)THEN
        .....      ! do whatever
        N.done=N.done+1
                   ! increment completed iterations
        ctrl=ctrl+N.incr
                   ! increment control variable
                   ! with saved 'incr'
        GOTO 1 ! loop
    ENDIF
```


- *ctrl* MUST NOT BE MODIFIED inside the loop
- exiting from loop, *ctrl* keeps its last value
- *fin, in* and *incr* can be modified, this does not affect the number of iterations nor the successive values of *ctrl*
- if a DO is within another DO, all of its range (up to the terminating label) must be inside it
- if a DO is within the THEN or ELSE part of an IF, it must completely contained in it
- ig a DO contain an IF..THEN..ELSE..ENDIF, the whole IF construct must be contained in the DO
- it is forbidden to jump into a loop from outside, but it is possible to jump out of a loop
- jumping to the terminal label of a loop (from inside) is allowed and starts next iteration


```
 N=100
 DO 10 I=1,N
C MISSING INCR MEANS 1
  READ *,X
  IF(X.LT.0)THEN
     N=N-1
     GOTO 10
  ENDIF
  WRITE(*,*)X,SQRT(X)
10 CONTINUE
 WRITE(*,*)N,' ITEMS PROCESSED'
```

Variant (nonstandard)

```
 DO I=1,10
....
 ENDDO
```

STOP
Blocks program execution (normal termination is at END of MAIN PROGRAM)

RETURN
Blocks subprogram execution and return to "calling program" (same effect as arriving to END of subprogram) See later.

The data structures

Only ONE: the ARRAY
ANY base type
1 to 7 dimensions
indexes are integers (any range of)
All arrays must be declared

```
DIMENSION A(10),M(-2:2)
```

A is 10 REAL elements, from **A(1)** to **A(10)**
M is 5 INTEGER elements from **M(-2)** to **M(2)**

Alternate style (recommended!)

```
REAL A(10)
INTEGER M(-2:2)
```

Never use DIMENSION!

```
REAL B(3,0:20)
```

63 REAL elements,
B(1,0), B(2,0), B(3,0),B(1,1)...B(3,20)

NOTE: stored in memory in THIS order

Warning about characters:
CHARACTER A*10, B(10)
Completely different! see CHARACTER*10 C(10)

✦array sizes must be constant!✦

```
parameter (n=10)
dimension a(n,20)
```


Operations on arrays

Almost none:

1) Input or output

2) pass to subroutines or functions as argument

3) take one element

(completely changed in FORTRAN 90!)


```
      PROGRAM TEST
      IMPLICIT NONE (A-Z)
      COMPLEX j
      PARAMETER (j=(0.,1.))
      REAL R,L,C,PI
      INTEGER I,N
      REAL OM(1000),T(1000),PHI(1000)
      COMPLEX Z(1000)
      PI=4*ATAN(1.0)
      READ(*,*)R,L,C
      DO 10 I=1,1000
        READ(*,*,END=11)OM(I)
10    CONTINUE
11    N=I-1
      DO 20 I=1,N
        Z(I)=R+j*(OM(I)*L+1/(OM(I)*C))
C     above inefficient unless compiler
C      very smart. Better:
C     Z(I)=CMPLX(R,OM(I)*L+1/(OM(I)*C))
        T(I)=ABS(Z(I))
        PHI(I)=ATAN2(IMAG(Z(I)),REAL(Z(I)))
C     atan2(x,y) means atan(x/y) but works
C     also if y=0
        PHI(I)=PHI(I)*180/PI
20    CONTINUE
      CALL MYPLOT(OM,T,PHI,1,N)
      OPEN(1,FILE='VALUE.DAT',
     $ FORM='UNFORMATTED')
      WRITE(1)N,R,C,L,OM,T,PHI
C     above WRITE quite poor!
      END
```


Part III: program units

Main:

```
PROGRAM PIPPO
...
END
```

Subroutine subprogram

```
SUBROUTINE S1(X,Y)
......
END
```

Function subprogram

```
INTEGER FUNCTION KLM(X,T)
C 'INTEGER' above can be omitted
C because of default rule:
C Initial I-N means integer
....
KLM=....
C This is the value the function
C RETURNS to the calling program
...
END
```

- main program 'called' from operating system
- main program 'calls' subprograms
- subprograms can call other subprograms
- NO RECURSION: subprograms cannot call directly or indirectly themselves

Using a Function

In expressions:

```
A=X*KLM(5.0, 3.4)
```

- The type of a function is determined like the type of any variable.
- Type of function in calling program INDEPENDENT from type of function in FUNCTION statement: definitions must agree!
- ***name1(name2)*** is interpreted as array element if *name1* declared as array, else interpreted as function call :
  - misspelled arrays cause calls to non-existent functions
  - function names must not be equal to variable, constant or array names

Using a subroutine

Usin the special statement CALL

```
CALL S1(3.2, ZETA)
```


Communicating with subprograms

Subprograms 'formal arguments'

```
      SUBROUTINE S1(X,T)
C declaration of formal arguments
      REAL X
      COMPLEX T(3)
C declaration of local variables
      COMPLEX S
      S=T(1)**2+T(2)**2+T(3)**2
      X=S
      END
```

Formal arguments used as normal variables inside subprograms

```
      PROGRAM M1
      COMPLEX A(3),B(3),X,Y
      ....
      CALL S1(X,A)
C X now contains real part of
C a(1)**2+a(2)**2+a(3)**2
      CALL S1(Y,B)
C same for b and y
      ....
```

- Actual arguments must match in number and type formal arguments. Only few compilers check!
- Constants can be used as actual arguments, but must not be used for a formal argument that is written by the subprogram

IN BOTH CASES: unpredictable results!


```
      CALL S1(X,Y)
      CALL S1(1.0,A)
```

Communicating with subprograms: the function return value

```
      DOUBLE PRECISION FUNCTION P(X,Z)
      DOUBLE PRECISION X,Z
      DOUBLE PRECISION X1
      X1=X
      P=1.
1     IF(X1.LT.Z)RETURN
       P=P*X1
       X1=X1-1
      GOTO 1
      END
      PROGRAM MAIN
      DOUBLE PRECISION P,X
C Note: declaration of P like
C declaration of X
      READ *,X
      WRITE(*,*)P(X,3.5D0)
C !ESSENTIAL WARNING:here 'D0' NEEDED
      END
```

- Return value indicated by function name
- Initially undefined (like any variable)
- Can be used as any variable
- Can be of any simple type (NOT array)
- Must be defined before reaching RETURN or END

Passing arrays to subprograms

Arrays can be formal and actual arguments

Portions of an array can be passed to a subprogram

```
      SUBROUTINE LIST1(X,N)
      REAL X(N)
      DO 1 I=1,N
       PRINT *,I,X(I)
    1 CONTINUE
      END
```

```
      SUBROUTINE LIST2(X,M,N)
      REAL X(M,N)
      DO 2 J=1,N
       DO 1 I=1,M
          PRINT *,I,J,X(I,J)
    1  CONTINUE
    2 CONTINUE
      END
```

```
      PROGRAM MAIN
      REAL X(3,3)
C in memory as : X(1,1) X(2,1) X(3,1)
C X(1,2) X(2,2) X(3,2) X(1,3) X(2,3)
C X(3,3)
      CALL LIST2(X,3,3)
C prints whole array as 2-dim
      CALL LIST1(X,9)
C prints whole array as 1-dim
      CALL LIST2(X,3,2)
C prints first 2 columms of X as 2-
  dim
C CALL LIST1(X(2,2),4)
C prints X(2,2) X(3,2) X(1,3) X(2,3)
C as 1-dim
      CALL LIST2(X(1,2),3,2)
C prints 2-nd and 3-rd columns of X
      END
```


1) Arrays can be formal arguments
2) **their dimension can be a constant or another formal argument**
   NOTE: local arrays can have ONLY constant dimensions
3) In correspondence with an array formal argument, one can pass either an array or an array element (of the same type as the formal argument!).
   An array element is interpreted as the first element of the 'formal array'. The operation is meaningful provided that the 'formal' array falls completely inside the 'actual' array

Example:

```
      DIMENSION X(10,10)
      CALL LIST1(X(1,4),50)
C OK (LIST1 uses X(1,4) up to
  X(10,8))
      CALL LIST1(X(1,7),50)
C WRONG (BUT NOT DETECTED BY THE
C COMPILER!): uses 20 elements beyond
C X(10,10) last "existing" object:
```


Communicating with subprograms: sharing variables


```
      PROGRAM MAIN
      PRINT *, FUN1(X), FUN2(X)
      END

      FUNCTION FUN1(X)
      LOGICAL INI
      DOUBLE PRECISION C(0:3)
      COMMON /A/INI,C
      IF(.NOT.INI)CALL INIT
      FUN1=C(0)+C(1)*X+C(2)*X**2+C(3)X**3
      END

      FUNCTION FUN2(X)
      LOGICAL INI
      DOUBLE PRECISION C(0:3)
      COMMON /A/INI,C
      IF(.NOT.INI)   CALL INIT
      FUN2=C(0)+SQRT(X)
      END

      SUBROUTINE INIT
      COMMON /A/INI,C
      LOGICAL INI
      DOUBLE PRECISION C(0:3)
      COMMON /A/INI,C
      C(0)=...
      C(1)=...
      C(2)=...
      C(3)=...
      INI=.TRUE.
      END
```

INI and C shared among all program units that contain the definition of 'common block /A/'

Allows sharing of memory outside a call path

Danger: only common block name globally known: definitions of variables contained in it MUST BE KEPT IDENTICAL **by hand** IN ALL THE INVOLVED PROGRAM UNITS

- Faster than using arguments passing
- VERY error-prone
- Often used as a DATA STRUCTURING facility
- Impossible to avoid

USE AT LEAST 'INCLUDE'

Data initialization

Problem:
How I know INI false at beginning?
I do not know:

Data initialization:

Local variables:

DATA *variable_name / value /*
DATA *list_of_names / list_of_values /*

```
SUBROUTINE ALFA
LOGICAL INI
REAL COF(20)
SAVE INI, COF
DATA INI , COF /.FALSE., 20*0.0/
....
IF(.NOT.INI)THEN
 CALL INITIALIZE(COF)
 INI=.TRUE.
ENDIF
.....
END
```

COMMON variables:

forbidden by the standard, allowed by most compilers. For the standard, use special subprograms:

```
BLOCK DATA INIT_A
LOGICAL INI
DOUBLE PRECISION C(0:3)
COMMON /A/INI,C
DATA INI /.FALSE./
END
```

- As many commons per BLOCK DATA subprogram as you want

Missing topics:

Passing subprograms as arguments
Static and automatic variables
Advanced I-O and formatting
Character handling
System interface
More control statements (at least COMPUTED GOTO)
...

Strengths of Fortran

Independent compilation
Array handling (in particular passing to subprograms)
COMPLEX (available to optimizers)
Large libraries
Close to hardware
Very efficient compilers

NEW FORTRAN (FORTRAN 90)
Data abstraction (very powerful)
Array constructs

FORTRAN RESOURCES AT ICTP

COMPILERS

PC's: Microsoft Fortran 5.1
FL
SUN workstations
f77
epcf90 (fortran 90 - available in a few days)
IBM workstations
f77
xlf (fortran 90 - to come)

LIBRARIES (only on UNIX machines)

SLATEC
CERN
NETLIB
see booklet

BOOKS:

Metcalf:
Effective Fortran 77

Wagener:
Fortran 77 Principles of Programming

Metcalf & Reid
Fortran 90 explained

ON-LINE DOCUMENTS for FORTRAN 90

(explore starting from "help")

Einarsson & Shokin
Fortran 90 for the Fortran77 Programmer
ftp://nsc.liu.se/pub/bibliotek/f77to90.txt

Metcalf
http://asis01.cern.ch/CN/CNTUT/f90/overview.html

Manchester University
http://www.hpctec.mcc.ac.uk/hpctec/courses/Fortran90/F90course.html

NAG
ftp://mycroft.plk.af.mil:/pub/Fortran_90/Tutorial/tutorial.tar.z