Erika Coppola from ESP section
E-mail: coppolae@ictp.it
Room 252, 2nd floor SISSA building

Alex Rodriguez (CM),Graziano Giuliani(ESP),
Uriel Morzan(CM), Rostami Samare(CM)

Computing-specific arrangements

You will need a ICTP UNIX account for this course.

USERNAME :
PASSWORD:

Book: any FORTRAN90 manual it is fine. In the library you can find this one that is ok "fortran 95/2003 explained" by Michel Metcalf, John Reid, Malcom Cohen, Oxford University Press

ICTP The Abdus Salam
International Centre
for Theoretical Physics

Filenames

In common with other operating systems, UNIX often names files with names of the form

\<filenamebody\>.\<extension\>

where the extension (usually no more than three or four letters) indicates the file type.

For example:

.txt : text file
.html (or .htm) : internet page file
.tex : document in the TeX (or LaTeX) typesetting language
.ps : postscript file

Filenames

Examples relevant to programming:

.f (or .f77) : source code file written in FORTRAN77

.f90 : source code file written in Fortran 90

.mod : Fortran module file

.c : source code file written in C

.cpp : source code file written in C++

.h : C/C++ header file

.o : object file (independent of language)

It's a good idea to stick to these conventions as programs may require them to identify file types.

Fortran Programming for Scientists

The course

- Objective: to learn to program Fortran 90/95 so that you

- can use a computer program to solve scientific problems.

- NO previous experience required

- NOT a computing course (so this course will not teach you to be a UNIX expert, or help you write device drivers, or develop a fancy graphical interface)

ICTP The Abdus Salam International Centre for Theoretical Physics

Purpose of the lectures themselves

If you already have programming experience, particularly if it's in FORTRAN, but even if it's in another language such as C or C++, then you may find the lectures largely Irrelevant, but we will go very fast and get to the point of really programming and learning useful programs for doing your own research.

The initial lectures are primarily designed for those **without** any previous programming experience. The lectures will start with the absolute basics of programming in FORTRAN90 and will take time to go through plenty of examples along the way.

Why computer programming?

- Computers do not get bored and do not make silly mistakes.

- Computers do arithmetic really quickly (desktop PC will do » 108 additions / multiplications per second)

- Computers are not clever ("garbage in, garbage out")

- So, if we need to invert a very large matrix, writing a computer program might help.

- But for performing symbolic integration, writing a computer program would be less helpful.

What is programming?

- A computer program is a set of instructions for the computer to execute sequentially.

- Eventually the program has to be translated into binary code for the processor to execute, but can originally be written in a choice of programming languages.

High-level languages

- High level language (such as Fortran, C, Pascal)

✓ Uses English-language keywords

✓ Is not dependent on a particular machine Architecture

✓ Requires significant interpretation by the compiler (see below) to turn the program into binary code

...
result = 0
do i = 1, 100
result = result + i
end do
...

Low-level languages

•Assembly language – a set of mnemonics that translate directly to instructions for the processor.

```
…
mov ax, 0
mov cx, 1
mov dx, 0x0a
label: startloop
add ax, cx
inc cx
cmp cx, dx
ifz goto startloop
…
```

From source code to binary executable

1. Initially the program source code is written in plain text by the programmer.

2. This file is then compiled (by an already existing application, called the *compiler*) to translate the plain text of the source code to binary code for the processor.

3. The compiler translates the text of the source code into a binary executable that is specific to a particular architecture. So I can write a Fortran program, it can be compiled by any Fortran compiler, but if I compile it for a Intel Pentium machine, then the resulting executable will certainly not work on a Sun machine.

Why Fortran?

- Fortran (FORmula TRANslator) was the first high-level programming language (1950s)

- Compilers are now so good that a program compiled from Fortran is almost as efficient as (i.e. as fast as and uses no more memory than) a program compiled from assembly language.

- Although competitors exist, Fortran is still the most widely used language for scientific problems

Why Fortran 90?

- Fortran 77 was for a long time the standard language for scientists, but had two major drawbacks:
  1. required fixed-form source code
  2. did not permit dynamic memory allocation

- Fortran 90 overcame these problems.

Preliminaries of syntax

Note: on this slide and henceforth, "Fortran" refers to "Fortran 90/95".
Fortran is case insensitive, so

x = mYvAr * mYoThervAr

is equivalent to

X = myVar * myOtherVar

A suggestion could be that UPPER CASE can be used for Fortran keywords, such as REAL, DO, END, and lower case for variables.

Preliminaries of syntax

A line in a Fortran program may be no longer than 132 characters

Anything following an exclamation mark (!) on a program line is a comment and is ignored by the processor. E.g.

```
y = 1 + x + x ** 2 / 2 ! first three terms of exp(x)
```

Spaces may be used freely (except within strings and tokens) to improve the layout of the program. E.g.

```
big_number = 1234 * 5678 ! Multiply two numbers and assign product
big_number=1234 * 5678 ! equivalent to line above
! big_number = 1 2 3 4 * 5678 ! SYNTAX ERROR: not allowed blanks
```

Variable names

A variable is an entity that is used to store a value, comparable to the use of variables in algebra.
A variable name in Fortran…

- must be between 1 and 31 characters in length

- must consist only of letters, numerals and underscores

- must begin with a letter

The following are acceptable variable names
```
x
my_variable
Result17
```

The following are not acceptable variable names

```
1x
```
doesn't begin with a letter

```
my$variable
```
'$' is not letter, numeral or underscore

```
my_really_really_long_variable_name
```
more than 31 characters

ICTP
The Abdus Salam
International Centre
for Theoretical Physics

Data types

Each variable in Fortran has a specific data type. The type of a variable is declared in a type declaration statement. Examples of the five data types intrinsic to Fortran are

```
INTEGER :: n ! signed integer
REAL :: x ! floating point number
COMPLEX :: impedance ! complex floating point
LOGICAL :: boolean_value ! takes either .true. or .false.
CHARACTER :: letter ! represents a single character
```

Note: there exist various types of integer, real, etc. This is covered later.

Basic operations

- **= assignment**
- **+ addition**
- **- minus and subtraction**
- **/ division**
- *** multiplication**
- **** raise to the power**
- **parentheses (…) can be used in the usual way**

Examples of basic assignments

```
x = 5.3           ! floating point assignment
n = 17 + 23       ! n is assigned the value 40
n = (12 + 1) * 3  ! use of parentheses
x = -y + 1        ! assuming y is another real
letter = 'q'      ! assigning a character
x = x + 1         ! increment the value of x
```

Basic I/O

- The abbreviation I/O stands for Input/Output

- Input: reading data from the user or from a storage device e.g. to a file on disk

- Output: writing data to the screen to e.g. a file on disk

- Fortran can automatically format data for you via "`PRINT *`" and "`READ *`"

Basic I/O: a sample program

```fortran
PROGRAM basic_io      ! start of program
  IMPLICIT NONE       ! assume nothing about variables
  REAL :: x           ! declare real variable

  ! Print a prompt
  PRINT*, 'Enter a real number:'
  ! Read in a number into the variable x
  READ*, x

  ! Print out what was entered
  PRINT*, 'You have entered the number ', x

  ! Print out the number and its square
  PRINT*, 'The square of ', x, ' is ', x * x

END PROGRAM basic_io !end of program
```
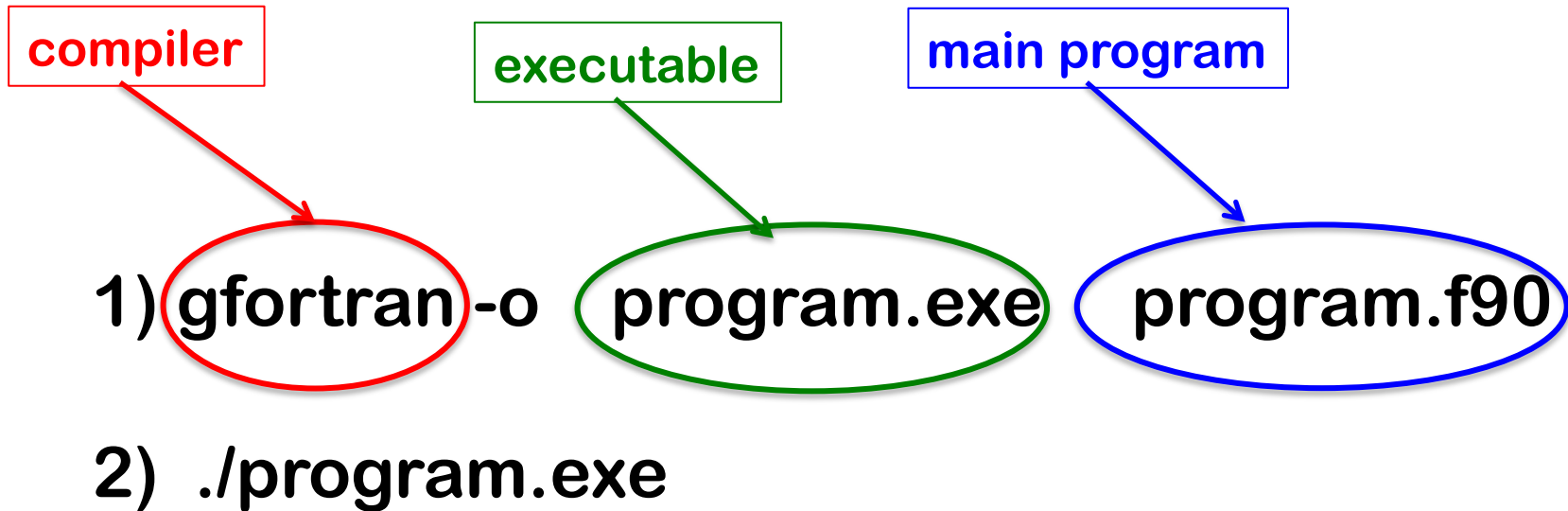
Our first program

```fortran
PROGRAM myfirstprogram
  IMPLICIT NONE      ! no assumption about variables
  REAL :: x, y, z     ! Declaration of x,y,z as real
  x= 5.1             ! Assign x
  y= -17.2           ! Assign y
  z= x*y             ! Assign (x*y) to z
  PRINT*, ' The product of x and y is ', z  ! Print out z
END PROGRAM myfirstprogram
```

How to compile

compiler

executable

main program

1) **gfortran -o program.exe program.f90**

2) ./program.exe

other compilers: ifort

Debugging? What is it?

The **primary** difference between a **programmer right out of college** and one with **five years' experience** is the **ability to debug programs**.

Debugging is still an art, not a science; but there are some techniques that can help.

"LISTEN, KID, HOW MANY TIMES DO I HAVE TO EXPLAIN IT TO YOU...BUGS DON'T BECOME PROGRAMMERS."

Debugging: the most classical episode

Student: My code isn't working. Can you tell me what's wrong with it?

Teacher: I'm not psychic! I need more information. :)

Can you answer these questions as best as you can, first?

1. What makes you say your code isn't working?
2. What did you expect your code to do and why?
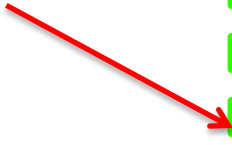3. What did your code do instead and how do you know?

Student: <answers questions>
Ohhh, I see what's wrong!

Teacher: Imagine if you had a habit of asking and answering these questions yourself. Half the time you'd solve your own problem and the other half you'd be able to ask a much more specific question and get relevant help more quickly.

Student: I totally agree.

Our first program: debugging

line 1 : **PROGRAM** myfirstprogram
line 2 : **IMPLICIT NONE**   ! no assumption about variables
line 3 : **REAL** :: x, y, z   ! Declaration of x,y,z as real
line 4 : x= 5.1   ! Assign x
line 5 : y= - ⎵ 17.2   ! Assign y
line 6 : z= x*y   ! Assign (x*y) to z
line 7 : **PRINT***, ' The product of x and y is ', z ! Print out z
line 8 : **END PROGRAM** myfirstprogram

error line 5

Our first program: debugging

line 1 : **PROGRAM** myfirstprogram
line 2 : **IMPLICIT NONE**   ! no assumption about variables
line 3 : REAL :: x, y, z   ! Declaration of x,y,z as real
line 4 : x= 5.1   ! Assign x
         print *, 'x',x,'y',y   ! debugging
line 5 : y= -17.2   ! Assign y
         print *, 'x',x,'y',y   ! debugging
line 6 : z= x*y   ! Assign (x*y) to z
line 7 : PRINT*, ' The product of x and y is ', z ! Print out z
line 8 : **END PROGRAM** myfirstprogram

error line 5

Operator precedence

- The arithmetic operators given above are evaluated in the conventional order of precedence: ** , * , / , - , + .

- When there are two operators of equal priority the operation proceeds from the left.

- Note therefore that x / y / z is equivalent to x / (y * z)

- It is advised that parentheses be used liberally in order to avoid confusion.

Operator precedence

```fortran
PROGRAM simple_math        ! start of program
   IMPLICIT NONE           ! assume nothing about variables
   REAL :: x, y, z         ! declare real variables
   INTEGER :: n            ! declare integer variable

   n = 2
   y = 2.1                 ! assign y
   z = -3.2                ! assign z

   x = y + z / 4.0         ! assign x
   PRINT*, x               ! print x

   x = (y + z) / 4.0       ! assign x again
   PRINT*, x               ! print x

   x = y ** n / z          ! x is y-squared over z
   PRINT*, x               ! print x

END PROGRAM simple_math ! end of program
```

Mixed-mode arithmetic

If the arguments to an arithmetic operator are

•of the same type, then the result is of the same type as the arguments.

•one of type **REAL** and the other of type **INTEGER** , then the **INTEGER** argument is converted to a **REAL** before the operation occurs.

Beware, therefore, dividing two integers, which (due to nonclosure) may result in truncation.

```fortran
PROGRAM integer_division ! start of program
  IMPLICIT NONE            ! assume nothing about variables
  PRINT*, 9 / 5            ! integer division !!
  PRINT*, 9.0 / 5.0        ! floating point division
END PROGRAM integer_division
```

WHERE to find the lectures:

/afs/ictp/public/c/coppolae/NUM2022