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

Sebastiano Pilati(CM)
Laura Mariotti (ESP),Ali Hassanali(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

Some UNIX commands

“ls” : lists files in a directory

“cd <dirname>” : change current directory to <dirname>

“cat <filename>” : dump the contents of <filename>

“rm <filename>” : delete (remove) file <filename>
!! USE WITH CAUTION !!

“less <filename>” : browse the contents of <filename>; cursor keys can be used, type ’q’ to exit

Working with NFS files

Files saved on the UITS central Unix computers Chrome, Cobalt, Zinc, Steel, EZinfo, and STARRS/SP are stored on the Network File Server (NFS). That means that your files are really on one disk, in directories named for the central Unix hosts on which you have accounts.

No matter which of these computers you are logged into, you can get to your files on any of the others. Here are the commands to use to get to any system directory from any other system:

cd /N/u/*username*/Chrome/
cd /N/u/*username*/Cobalt/
cd /N/u/*username*/Zinc/
cd /N/u/*username*/Steel/
cd /N/u/*username*/Ezinfo/
cd /n/u/*username*/SP/

Be sure you use the capitalization just as you see above, and substitute your own username for *username*.

For example, if Jessica Rabbit is logged into her account on Steel, and wants to get a file on her EZinfo account, she would enter:

cd /N/u/jrabbit/Ezinfo/

Now when she lists her files, she'll see her EZinfo files, even though she's actually logged into Steel.

You can use the ordinary Unix commands to move files, copy files, or make symbolic links between files. For example, if John Doe wanted to move "file1" from his Steel directory to his EZinfo directory, he would enter:

mv -i /N/u/jdoe/Steel/file1 /N/u/jdoe/Ezinfo/

This shared file system means that you can access, for example, your Chrome files even when you are logged into Cobalt, and vice versa. However, if you are logged into Chrome, you can only use the software installed on Chrome —only users' directories are linked together, not system directories.

b017.9808

Unix commands reference card

Abbreviations used in this pamphlet

Ctrl/x	hold down control key and press x
d	directory
env	environment
f	filename
n	number
nd	computer node
var	variable
[y/n]	yes or no
[]	optional arg
...	list

August 1998

To access this guide on the
World Wide Web, set your browser to
http://www.indiana.edu/~uitspubs/b017/

INDIANA UNIVERSITY

Environment Control

Command	Description
cd *d*	*Change to directory d*
mkdir *d*	*Create new directory d*
rmdir *d*	*Remove directory d*
mv *f1 [f2...] d*	*Move file f to directory d*
mv *d1 d2*	*Rename directory d1 as d2*
passwd	*Change password*
alias *name1 name2*	*Create command alias*
unalias *name1*	*Remove command alias name1*
rlogin *nd*	*Login to remote node*
logout	*End terminal session*
setenv *name v*	*Set env var to value v*
unsetenv *name1 name2...]*	*remove environment variable*

Output, Communication, & Help

Command	Description
lpr -P *printer f*	*Output file f to line printer*
script *[f]*	*Save terminal session to f*
exit	*Stop saving terminal*
session	
mail *username*	*Send mail to user*
biff *[y/n]*	*Instant notification of mail*
man *name*	*UNIX manual entry for*
name	
learn	*Online tutorial*

Process Control

Command	Description
Ctrl/c *	*Interrupt processes*
Ctrl/s *	*Stop screen scrolling*
Ctrl/q *	*Resume screen output*
sleep *n*	*Sleep for n seconds*
jobs	*Print list of jobs*
kill [%n]	*Kill job n*
ps	*Print process status stats*
kill -9 *n*	*Remove process n*
Ctrl/z *	*Suspend current process*
stop %*n*	*Suspend background job n*
command&	*Run command in background*
bg [%*n*]	*Resume background job n*
fg [%*n*]	*Resume foreground job n*
exit	*Exit from shell*

Environment Status

Command	Description
ls *[d] [f...]*	*List files in directory*
ls -l *[f...]*	*List files in detail*
alias *[name]*	*Display command aliases*
printenv *[name]*	*Print environment values*
quota	*Display disk quota*
date	*Print date & time*
who	*List logged in users*
whoami	*Display current user*
finger *[username]*	*Output user information*
chfn	*Change finger information*
pwd	*Print working directory*
history	*Display recent commands*
! n	*Submit recent command n*

File Manipulation

Command	Description
vi [f]	*Vi fullscreen editor*
emacs *[f]*	*Emacs fullscreen editor*
ed *[f]*	*Text editor*
wc *f*	*Line, word, & char count*
cat *f*	*List contents of file*
more *f*	*List file contents by screen*
cat *f1 f2 > f3*	*Concatenates f1 & f2 into f3*
chmod *mode f*	*Change protection mode of f*
cmp *f1 f2*	*Compare two files*
cp *f1 f2*	*Copy file f1 into f2*
sort *f*	*Alphabetically sort f*
split *[-n] f*	*Split f into n-line pieces*
mv *f1 f2*	*Rename file f1 as f2*
rm *f*	*Delete (remove) file f*
grep *'ptn' f*	*Outputs lines that match ptn*
diff *f1 f2*	*Lists file differences*
head *f*	*Output beginning of f*
tail *f*	*Output end of f*

Compiler

Command	Description
cc *[-o f1] f2*	*C compiler*
lint *f*	*Check C code for errors*
f77 *[-o f1] f2*	*Fortran77 compiler*
pc *[-o f1] f2*	*Pascal compiler*

Press RETURN at the end of each command, except those marked by an asterisk (*).

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)

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 PASCAL, 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 FORTRAN 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
my$variable
my_really_really_long_variable_name
```

doesn't begin with a letter

'$' is not letter, numeral or underscore

more than 31 characters

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
```

Our first program

```
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
```

Compilers available:

1) gfortran
2) ifort
3) pgf90

How to compile :

1) pgf90 -o program.exe program.f

2) ./program.exe

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

```
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
```

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

```
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.

```
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/NUM2014