

PHYS-4007/5007: Computational Physics
Course Lecture Notes
Section II

Dr. Donald G. Luttermoser
East Tennessee State University

Version 5.0

Abstract

These class notes are designed for use of the instructor and students of the course **PHYS-4007/5007: Computational Physics** taught by Dr. Donald Luttermoser at East Tennessee State University.

II. How To Select a Programming Language

A. Which is the Best Programming Language for Your Work?

1. Selection of a programming language for a given task can be a daunting task. One must weight the availability of the language, the functions it is capable of, and the speed and debugging capabilities offered by the compiler.
 - a) Different fields of study typically have their favorite programming language.
 - b) Many mathematicians use **Mathematica**, **Maple**, and **Mathlab**.
 - c) In physics and astronomy, most of the large codes are written in **Fortran 77**, with the remaining small percentage written in either the newer **Fortran 90** or **C** (very few are written in **C++**).
 - d) Astronomers who use a lot of NASA space data typically use the **Interactive Data Language (IDL)** for their work.
 - e) Since we don't have time to cover each and every programming language on the market, we will compare the three most commonly used in physics and astronomy: **Fortran 77**, **C**, and **IDL**.
 - i) The details of **Fortran 77** and an overview of **Fortran 90/95** can be found in Appendix A of these notes.
 - ii) The details of **C** and an overview of **C++** can be found in Appendix B.

- iii) The details of IDL can be found in the next section (*i.e.*, §III) of these notes.
 - f) Any coding introduced in these notes will be written in either IDL or Fortran 77, hence Appendix A and §III should be studied carefully so you fully understand what is being written in these code segments.
2. Most programming languages have analogous elements in their programming style, which can be reduced to 8 sections as shown in Table II-1.
 3. No matter which programming language you choose, you need to come up with a style and be consistent throughout the code.
 - a) You do this not only to make it easier for other users to figure out what you are doing, it makes it easier on you when you go back to a code after you haven't looked at in a long time.
 - b) **Always have numerous comments throughout the code describing what is being done!**
 - c) For my style, I use lower-case letters (except for the first letter of words at the beginning of a sentence) for comments and upper-case letters for the coding itself. (Though in Table II-1, I use lower-case letters for demonstration purposes only.)
 - d) Try to use a lot of subroutines (or procedures) and functions \implies make your code as modular as possible. (PANDORA has over 5000 subroutines in it!)

Table II-1: Analogous Elements in Fortran, C, and IDL

Fortran	C	IDL
Program Structure		
program f	main()	(no main program structure)
function f(x)	double f(double x)	function f, x
subroutine f(x)	void f(x)	pro f, x
Operations		
x**y	pow(x,y);	x^y
x = y	x = y;	x = y
do 2 k=1,kmax,1	for (k=1;k<=kmax;k++)	for k=0,kmax-1,1 do
do 2 k=kmax,1,-1	for (k=kmax;k>=1;k-)	for k=kmax-1,0,-1 do
Data Type Declarations		
real*8 x, y	double x, y;	x = 0.0d0 & y = 0.0d0
integer i, j	int i, j;	i = 0 & j = 0
real*8 y(100,100)	double y [100] [100];	y = dblarr(100,100)
integer ii(100,100)	int ii [100] [100];	ii = intarr(100,100)
data mn/0.0/	define mn 0.0;	defsysv, '!mn', 0.0, /read_only
character jan	char jan;	jan = '' (NULL string)
Input and Output to Screen		
read(*,*) x1	scanf("%lf", &x1);	read, x1
write(*,*) 'Enter radius:'	printf("Enter radius:");	print, 'Enter radius:'
write(*,*) 'radius = ', radius	printf("radius = %f", radius);	print, 'radius = ', radius
Input and Output to Files		
	FILE *fout;	
open(7,file='x.dat',status='new')	out = fopen("x.dat", "w");	openw, 7, 'x.dat'
write(7,'(f10.2)') Din	fprintf(fout, "%lf", Din);	printf, 7, Din, format='(f10.2)'
read(7,100) Din	fscanf(fin, "%lf", &Din);	readf, 7, Din
Control Structure: if		
if (den .eq. 0) then	if (den == 0) {	if den eq then begin
write(*,*) 'Trouble'	printf("Trouble");	print, 'Trouble'
endif	}	endif
Control Structure: if ... else		
if (den .eq. 0) then	if (den == 0)	if den eq then begin
x = 0	{ x = 0; }	x = 0
else	else	endif else begin
x = x1/den	{ x = x1/den; }	x = x1/den
		endelse

Table II-1: (cont.) **Analogous Elements in Fortran, C, and IDL**

Fortran	C	IDL
Control Structure: for		
do 2 k=1,kmax,1 term = r*term 2 continue	for (k=1;k<=kmax;k++) { term = r*term; sum = sum+term; }	for k=0,kmax-1,1 do begin term = r*term endfor
Switch		
goto (1,2), choice 1 series = first 2 series = second	switch (choice) { case 1: series = first; break; case 2: series = second; break; }	case choice of 1: series = first 2: series = second else: endcase

B. The Fortran Programming Language.

1. Fortran was first developed in 1957 by IBM and was the first *high-level* programming language.
 - a) Its name is derived from “FORmula TRANslation.”
 - b) Fortran has evolved considerably since 1957 \implies Fortran I \rightarrow Fortran II \rightarrow Fortran III \rightarrow Fortran 66 (Fortran IV) \rightarrow Fortran 77 (Fortran V) \rightarrow Fortran 90 \rightarrow Fortran 95 \rightarrow Fortran 2003 \rightarrow Fortran 2008. Note that each new version typically can compile code all the back to Fortran 77.
 - c) Fortran is a **sequential** programming language — there is a logical flow to the coding: the first line of the code is operated upon first, the 2nd line, second, through the last line, last.
 - d) Most large *number-crunching* codes in science are written in Fortran 77 which came on the market in 1978.

2. Fortran has a number of features that make it useful to the scientific community:
 - a) **Portability:** Portability means that it is easy to move from one machine to another. While no large program can move from one machine to another totally without difficulty, problems are usually minor, provided the Fortran 77 standard has been strictly adhered to.
 - b) **Availability:** Fortran is probably more widely used than any other programming language in the physics and astronomy community. Compilers are typically available for all machine types and operating systems likely to be encountered in a university.
 - c) **Efficiency:** Partly because of its simple, static use of store, and the un-complicated nature of its constructions, and partly because of a massive investment in compiler development, numerically intensive programs written in Fortran are generally faster than those in any other high-level language. The technique of *optimization*, by which a compiler alters the machine code it generates (without changing the end result!) to speed the calculation is well developed in Fortran.
 - d) **Fortran Libraries:** Very considerable effort has gone, over many years, into the production of libraries of routines that may be called from Fortran programs. Chief amongst these are libraries of numerical-analysis routines (such as BLAS, LAPACK, and NAG libraries) and routines to do graph plotting (such as the GINO and the now defunct DI-3000 libraries).

3. Fortran is designed to be *modular*:
 - a) One has a main program that drives the code and calls **subroutines** and **functions**.
 - b) A good Fortran code typically has a short main program which call many subroutines.
 - c) Subroutines and functions can call other subroutines and functions.

C. Running Fortran in Linux.

1. To write a Fortran code under Linux, use the following recipe:
 - a) Change directories to whatever subdirectory you want as your working directory.
 - b) Let's say we wish to create a code called `atlas.f` (or we already had one by that name). Issue the command (at the Unix prompt):

```
emacs atlas.f
```
 - c) This will bring up the `emacs` editor window and start writing your code. When done, click the "Save" button (followed by the "Exit" button if you don't need to make any more modifications). Let's say that we edit a second file which contains some operating system specific command (like the `GETNODE()` function that is available on some Fortran compilers) called `atlasunix.f`.
2. The Linux operating system uses the latest GNU Fortran compiler called `gfortran`. Read the 'man' pages (*e.g.*, `man gfortran`) for a listing of compiler options. This Fortran compiler can create executable programs from files written in either Fortran 77, Fortran 90, and Fortran 95.

3. Compiling and linking the code are performed with one command (again at the Unix prompt):

```
gfortran -o atexe atlas.f atlasunix.f
```

- a) Here the “-o” flag tells the linking software to save the executable code in a file called `atexe` (if ‘-o’ was not included, the executable would be saved in a file named `a.out`).
- b) By default, the `gfortran` compiler will issue floating point overflows and underflows warnings (see Appendix A).

D. The C Programming Language.

1. C, like Fortran 77, is referred to as a **sequential** programming language \implies the flow of the operations carried out by the computer line by line from the beginning of the main program to the end of it.
2. Like Fortran, C is considered a higher-level language, but it has some lower-level language functionality \implies it can access the operating system much easier and more efficiently than Fortran 77/90.
 - a) A “C” program = functions (executable code) + variables (data).
 - b) Every program must have a function called `main`, execution starts there. `main` may call other functions.
 - c) C has no concept of a **Program** as in Fortran, just functions.
 - d) The C view is that `main` is called by the operating system; the operating system may pass arguments to `main`, as well

as receive return values (*e.g.*, return 0;); however, above, we choose to make `main` take no arguments – (void).

- e) `/* ... */` is a comment and is ignored by the compiler; the C standard says that comments cannot be nested. Also, comments **must** be terminated explicitly, *i.e.*, new-line does not terminate them — this is a common source of compiler errors, and, for the unwary, can be very difficult to trace. It’s a good idea for every C program to have a header comment containing:
- i) Name of the program — to correspond to the filename.
 - ii) Authors name.
 - iii) Date.
 - iv) Brief indication of function and purpose of the program, *e.g.*, assignment number, or project, and what it does.
3. Appendix B of these course notes contains some details and a tutorial in programming with the C language. One thing to note is that C is not designed to be a “number-crunching” programming language. It has very limited mathematical operators and one must load additional math libraries at compile time to do any math beyond simple arithmetic. If you are going to be *crunching numbers*, then Fortran is typically the code of choice.