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

Dr. Donald G. Luttermoser
East Tennessee State University

Version 7.1

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.

I. An Introduction to Scientific Computing

A. Computer Types.

1. With the widespread availability of fast computers at affordable prices, it has become important to understand the capabilities and details of numerical modeling.

2. In physics and astronomy, many talk about five different classes or types of computer hardware. Over time, some of these classes have become extinct. From most to least expensive they are:

a) **Supercomputers:** A mainframe with many 64-bit CPU chips — **vector processors**. Cray used to be the biggest maker of supercomputers, but they have now gone out of business due to the speed and low-cost of the PCs and the advent of computer clusters. There are still a few Crays in operation around the country.

b) **Computer Clusters:** Since the early 21st century, small "cluster" machines, some with multi-CPU chips (in a single box) and others with multi-computers (in a single box with one or two CPUs) connected via a high speed network, have taken the place of "supercomputers" due to their much lower cost.

c) **Mainframes:** These were large computers (large photocopier size or larger) which allowed for multi-user use from remote locations via a network. These machines all had multi-user operating system running the architecture.

i) Here "multi-user" means many operators using the machine at the *same time*.

ii) Mainframes in the 1960s and 1970s typically had 32-bit CPU chips.

iii) Machines in the 1970s and 1980s contained 64-bit CPU chips. The old VAXes (various models, 750, 780, 8600, etc.) and the IBM 360 are two examples of mainframes.

iv) These machines became extinct by the late 1990s due to the popularity of multi-user workstations and fast, cheap PCs (see below).

d) **Workstations:** A machine that runs a *multi-user* operating system like Unix and its variants. They are essentially scaled down versions of the older mainframe computers.

i) They typically have a 64-bit CPU processor (or series of processors), though some older machines (pre-1995) only have 32-bit chips in them.

ii) On these machines, numerous users can operate the machine remotely (and at the computer console) at the same time.

iii) Examples of such machines are the Sun workstations, AlphaStations (originally made by the now defunct Digital Equipment Company), and Silicon Graphics workstations.

iv) Note that the PC world has been calling machines running various flavors of Microsoft Windows "workstations." However, these machines are not workstations from the definition above, since they

are not designed for simultaneous, real-time, multi-users use.

v) Workstations are now virtually extinct due to the advent of much cheaper fast PCs.

e) **Personal Computers (PCs)**: Typically a 64-bit CPU processor (or series of processors) running an *operating system* (OS) of either some version of Microsoft Windows, Linux (with X-Windows), or Mac OS (on the MacIntosh's).

i) Note that there are still many of the older 32-bit PCs being used by various people.

ii) These machines, both the 32-bit and 64-bit CPUs, are designed for *single-user* mode (*i.e.*, one user at a time).

3. The *size* of the CPU can be as important as the *clock speed* of the CPU in terms of computer speed.

a) Early PCs only had only an 8-bit CPU chip. They evolved to 16-bit, then 32-bit, and now 64-bit.

b) There are two reasons why the size of the CPU is important.

i) The maximum and minimum size of a number is limited by the *chip size*.

ii) The amount of precision possible for a number is also limited by the CPU size (though some compilers are sophisticated enough to increase precision for a given chip beyond what the chip can normally provide).

B. Operating Systems

1. Software designed to control the interface between a user or computer programs and the CPU of a computer is known as an **Operating System**.

a) A simple definition of an operating system is the suite of programs that make the hardware usable. The operating system manages the CPU, disks, and I/O devices.

b) Manipulation of the operating system was typically one of the hardest aspects of learning to use computers which is why the "GUI (Graphic User Interface) mentality" has taken hold of the modern computers — the user no long needs to talk to the operating system, the GUI does it for you!

2. Some of the most popular operating system in use today are Microsoft Windows, Mac OS, and Linux.

a) Since most users now-a-days use Windows machines or Macs, both with 'front-end' GUIs, we will not go into the details of these operating systems.

b) Many scientists in Astronomy and Physics prefer using Unix/Linux-based machines to carry out their research. Note that we will be primarily be using the Linux operating system in this class.

c) Appendix A has a brief review of the history of operating systems, please review this appendix at your earliest convenience.

d) In addition, **carefully** review Section C of this Appendix A to learn the structure of Unix/Linux and learn some of the

more common operating system commands used in these types of operating systems.

e) Note that one communicates with the Linux operating system by opening a Terminal window on the display screen. Instructions for how to do this will be passed out in class.

C. Programming Languages

1. When scientists need to use computers to help them carry out their research, they need to decide on the best operating system to use and the most suitable programming language to use to carry out this research.

a) Data analysis research will typically require software that can produce graphics in addition to performing mathematical calculations. In Physics and Astronomy, it is useful for a software package to have functions capable of fitting data, such as, least-squares routines to fit straight lines to data and various other curve-fitting routines (*e.g.*, CURVEFIT, GUASSFIT, REGRESS, etc.)

b) Numerical modeling will typically require a programming language that has a rich set of mathematical function routines such as trigonometry functions (*e.g.*, SIN, COS, TAN, etc.), logarithms and exponentiation (*e.g.*, LOG, LOG10, EXP, etc.), integration routines (*e.g.*, QSIMP, INT_2D, INT_3D, etc.), and matrix manipulation (*e.g.*, CRAMER, DETERM, INVERT, NORM, etc.).

2. 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 languages.

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 90/95/2000, C, or Python (note that very few are written in C++).

i) Yes, that is correct, the decades old Fortran 77 is still in wide use in physics and astronomy.

ii) Mainly this is due to the fact that their are numerous libraries of Fortran 77 codes that are available — Fortran is designed to be a number-crunching programming language, and finally, few want to rewrite codes with thousands of subroutines and hundreds of thousands lines of code.

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 focus on the three most commonly used in physics and astronomy: Fortran 77, Python, and IDL.

i) An introduction to programming in these three languages can be found in the next section (§II) of these notes.

ii) The details of Fortran 77 and an overview of Fortran 90/95/2000 can be found in Appendix B of

these notes.

iii) The details of Python can be found in Appendix C.

iv) An introduction to programming in IDL can be found in Appendix D of these notes.

v) The details of C and an overview of C++ can be found in Appendix E if students are interested in learning these programming languages.

f) 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.

d) Try to use a lot of subroutines (or procedures) and functions $\Longrightarrow$ make your code as modular as possible. (The radiative transfer code PANDORA has over 5000 subroutines in it!)

D. Data Types

1. There are two basic types of data that are operated on and stored by computers $\Longrightarrow$ **integers** and **real numbers**. Depending on the programming language one is using, there are various types of "integers" and "reals." The details of how computers store and operate on these data types are given in Appendix F on "How Computers 'See' Numbers and Letters." You should review this appendix before reading this subsection of the notes.

2. **Integers** come in a variety of flavors based upon how many bits are used to store these numbers.

a) **Logical**: 1-bit word, values = [.FALSE. (=0) : .TRUE. (=1)], maximum value = 1.

b) **Short Integer**: 8-bit or one-byte word length, range = [–128 : 127], maximum value = $2^7 - 1$, number of digits = 3.

c) **Character**: Like a short integer typically containing the ASCII code of the character. Note that in higher-level programming languages (like Fortran, Python, and IDL), characters are stored and operated upon using **string** notation (characters surrounded by single- and/or double-quotation marks). The compiler changes these characters to the short integer ASCII code, then to binary, to which the machine then operates upon. The next subsection covers the ASCII character set.

d) **Integer**: 16-bit or two-byte word length, range = [–32,768 : 32,767], maximum value = $2^{15} - 1$, number of digits = 5.

e) **Long Integer** (sometimes just called 'Long' or 'Double-Precision Integer'): 32-bit or four-byte word length, range = [–2,147,483,648 : 2,147,483,647], maximum value = $2^{31}-1$, number of digits = 10.

3. Like integers, **reals** come in a variety of precision flavors based upon how many bits are used to store these numbers.

a) **Single-Precision Real** (sometimes called *Floats* or *Real∗4* numbers): 32-bit = 4-byte word-size numbers, 6-7 decimal places of precision (1 part in 2^{23}), and a range = $[1.17549435 \times 10^{-38} : 3.40282347 \times 10^{38}]$.

b) **Double-Precision Real** (sometimes called just *Double Precision* or *Real∗8*): 64-bit (2 32-bit words → 11 bits used for the exponent, 1 for the sign, and 52 bits for the mantissa) = 8-byte word-size numbers, about 16 decimal places of precision (1 part in 2^{52}), and a range = $[2.2250738585072014\times10^{-308} : 1.7976931348623157\times10^{308}]$.

c) **Quad-Precision Real** (*Real∗16*): 128-bit (4 32-bit words → 15 bits used for the exponent, 1 for the sign, and 112 bits for the mantissa) = 16-byte word-size numbers, about 36 decimal places of precision (1 part in 2^{112}), and a range = $[\sim3.362\times10^{-4932} : \sim1.189\times10^{4932}]$. Currently this precision is only available in Fortran compilers on some 64-bit workstations and clusters.

d) **Complex Numbers**: Some programming languages (in particular, Fortran and IDL) have complex data types. A complex number has the form

$$z = x + iy \ , \qquad \text{(I-1)}$$

where $i \equiv \sqrt{-1}$:

i) x corresponds to the real part of the number.

ii) y corresponds to the imaginary part.

iii) These numbers are stored as two-element arrays (real, imaginary) of real numbers.

E. Understanding ASCII Format. (IMPORTANT SECTION)

1. English letters and punctuation marks are processed on most computers as an 8-bit number called **ASCII** (American Standard Code for Information Interchange) format (note that the first "sign" bit is not used in ASCII protocol, hence it is actually a 7-bit number).

a) Each character on the American keyboard (including their capital representations) has an ASCII numeric-value associated with it, ranging from 0 to 127 (*e.g.*, '0' (zero) has an ASCII value of 48, 'A' (capital-a) = 65, and 'a' = 97, see Table I-1 for a complete list ASCII numeric values).

b) The first 32 characters in the ASCII-table (0 – 31) are unprintable control codes and are used to control peripherals such as printers.

c) I will often refer to "text" ('*.txt') files (in the Microsoft world), that is, those files that can be printed or viewed with a text editor, such as Notepad, vi, or Emacs, as 'ASCII' files (as they are called in the Unix/Linux world). **Note that Microsoft Word is <u>not</u> a text editor (see below).**

2. Note that there is also an *extended ASCII table* of values which uses all 8-bits to represent a character corresponding to *non-American* keyboard letters ranging in values from 128 to 255.

Table I–1: The ASCII Table

Decimal Value	Binary Value	Symbol	Name Description
0	00000000	NUL	Null character
1	00000001	SOH	Start of Heading
2	00000010	STX	Start of Text
3	00000011	ETX	End of Text
4	00000100	EOT	End of Transmission
5	00000101	ENQ	Enquiry
6	00000110	ACK	Acknowledgment
7	00000111	BEL	Bell
8	00001000	BS	Back Space
9	00001001	HT	Horizontal Tab
10	00001010	LF	Line Feed
11	00001011	VT	Vertical Tab
12	00001100	FF	Form Feed
13	00001101	CR	Carriage Return
14	00001110	SO	Shift Out / X-On
15	00001111	SI	Shift In / X-Off
16	00010000	DLE	Data Line Escape
17	00010001	DC1	Device Control 1 (oft. XON)
18	00010010	DC2	Device Control 2
19	00010011	DC3	Device Control 3 (oft. XOFF)
20	00010100	DC4	Device Control 4
21	00010101	NAK	Negative Acknowledgment
22	00010110	SYN	Synchronous Idle
23	00010111	ETB	End of Transmit Block
24	00011000	CAN	Cancel
25	00011001	EM	End of Medium
26	00011010	SUB	Substitute
27	00011011	ESC	Escape
28	00011100	FS	File Separator
29	00011101	GS	Group Separator
30	00011110	RS	Record Separator
31	00011111	US	Unit Separator
32	00100000		Blank space
33	00100001	!	Exclamation mark
34	00100010	”	Double quotes (or speech marks)
35	00100011	#	Number symbol
36	00100100	$	Dollar symbol
37	00100101	%	Percent symbol
38	00100110	&	Ampersand symbol
39	00100111	’	(closing) Single quote

Table I–1: The ASCII Table (continued)

Decimal Value	Binary Value	Symbol	Name Description
40	00101000	(	Open parenthesis
41	00101001	)	Close parenthesis
42	00101010	*	Asterisk symbol
43	00101011	+	Plus sign
44	00101100	,	Comma
45	00101101	–	Hyphen
46	00101110	.	Period
47	00101111	/	Slash or divide symbol
48	00110000	0	Zero
49	00110001	1	One
50	00110010	2	Two
51	00110011	3	Three
52	00110100	4	Four
53	00110101	5	Five
54	00110110	6	Six
55	00110111	7	Seven
56	00111000	8	Eight
57	00111001	9	Nine
58	00111010	:	Colon
59	00111011	;	Semicolon
60	00111100	<	Less than (or open angled bracket)
61	00111101	=	Equals sign
62	00111110	>	Greater than (or close angled bracket)
63	00111111	?	Question mark
64	01000000	@	At symbol
65	01000001	A	Uppercase A
66	01000010	B	Uppercase B
67	01000011	C	Uppercase C
68	01000100	D	Uppercase D
69	01000101	E	Uppercase E
70	01000110	F	Uppercase F
71	01000111	G	Uppercase G
72	01001000	H	Uppercase H
73	01001001	I	Uppercase I
74	01001010	J	Uppercase J
75	01001011	K	Uppercase K
76	01001100	L	Uppercase L
77	01001101	M	Uppercase M
78	01001110	N	Uppercase N
79	01001111	O	Uppercase O

Table I–1: The ASCII Table (continued)

Decimal Value	Binary Value	Symbol	Name Description
80	01010000	P	Uppercase P
81	01010001	Q	Uppercase Q
82	01010010	R	Uppercase R
83	01010011	S	Uppercase S
84	01010100	T	Uppercase T
85	01010101	U	Uppercase U
86	01010110	V	Uppercase V
87	01010111	W	Uppercase W
88	01011000	X	Uppercase X
89	01011001	Y	Uppercase Y
90	01011010	Z	Uppercase Z
91	01011011	[	Opening square bracket
92	01011100	\	Backslash
93	01011101	]	Closing square bracket
94	01011110	^	Caret - circumflex
95	01011111	_	Underscore
96	01100000	‘	Grave accent (opening single quote)
97	01100001	a	Lowercase a
98	01100010	b	Lowercase b
99	01100011	c	Lowercase c
100	01100100	d	Lowercase d
101	01100101	e	Lowercase e
102	01100110	f	Lowercase f
103	01100111	g	Lowercase g
104	01101000	h	Lowercase h
105	01101001	i	Lowercase i
106	01101010	j	Lowercase j
107	01101011	k	Lowercase k
108	01101100	l	Lowercase l
109	01101101	m	Lowercase m
110	01101110	n	Lowercase n
111	01101111	o	Lowercase o
112	01110000	p	Lowercase p
113	01110001	q	Lowercase q
114	01110010	r	Lowercase r
115	01110011	s	Lowercase s
116	01110100	t	Lowercase t
117	01110101	u	Lowercase u
118	01110110	v	Lowercase v
119	01110111	w	Lowercase w

Table I–1: The ASCII Table (continued)

Decimal Value	Binary Value	Symbol	Name Description
120	01111000	x	Lowercase x
121	01111001	y	Lowercase y
122	01111010	z	Lowercase z
123	01111011	{	Opening brace
124	01111100	—	Vertical bar
125	01111101	}	Closing brace
126	01111110	∼	Equivalency sign – tilde
127	01111111		Delete

Important Note: Most text editors will not recognize the extended ASCII table. As such, you should never use them when writing computer programs or LaTeX documents. If you don't see the character on a standard American keyboard, then you should not include that symbol (such as π or Å) in your program. See http://www.asciitable.com/ to see the extended ASCII code numeric values.

3. When you are writing code or a LaTeX document in a text editor, **do not simply cut and paste from text that you might see on the web or in a Microsoft Word document**.

a) Just because your eye sees a standard American keyboard character in such a document, it does not mean it is ASCII text.

b) Even if the editor you are using does print the character as you see it in the document, it doesn't mean there is not unseen control characters included with the character (*i.e.*, a 'font' control character or an 'underline' control character). Only clip and paste from know ASCII files (*e.g.*, a file opened with the Microsoft Notepad editor to a file opened with the Emacs editor).

c) Once you run a code through a compiler, such as Fortran or IDL, you may get errors indicating an unrecognized symbol in your code, then the character you see in the editor is not an ASCII character.

d) If a symbol you cut and paste (say the Greek π symbol) from the web shows up in the editor but does not appear in the final PDF document after compiling through LaTeX, then the character you see in the editor is not an ASCII character and should not be used. For instance, in LaTeX, you create the pi symbol in the PDF output file by typing \$\pi\$ in your LaTeX document (*e.g.*, a file named 'mydocumet.tex') — this is how the pi symbol above was created in this PDF document. See §III for details on document preparation using LaTeX.

e) Finally note that Microsoft Word is a document and word processing software, not a text editor. **Do not write computer code or LaTeX files using Microsoft Word, instead use a text editor such as Emacs or Notepad.**

F. The Inner Workings of a Computer.

1. A computer does mathematics (which is the same thing as running programs, whether your program is designed to do math problems or not) via **machine language**.

a) Machine language is built into the CPU of the machines — each CPU family of chips has its own specific machine language or *architecture*, where machine language manipulates numbers on the "bit" level. Coding is written in machine language which controls the CPU registers.

i) The old 32-bit Intel Pentium chips used one type of machine language, primarily based on the *CISC*

(Complex Instruction Set Computer). Such a processor uses small code sizes carrying out instructions over multi-clock cycles. In this architecture, transistors on a chip are used for storing complex instructions.

ii) Meanwhile the 64-bit chip "workstation-class" machines used their own machine language based on the *RISC* (Reduced Instruction Set Computer) architecture. Such a processor uses large code sizes carrying out instructions over a single clock cycle. This architecture spends more transistors on memory registers than the CISC chips.

iii) Newer chip architectures have been developed in the recent past especially for *cluster* machines with multiple CPUs, such as the *massive parallel processing* (MPP) architecture.

b) Operating systems are designed to communicate with the machine language of the chip. As such, a discussion of computer hardware should include a discussion of operating systems which is covered in Appendix A of these notes. However, first we need to define a few terms about the inner workings of a computer. (Refer to Chapter 13 in your textbook.)

i) **CPU:** The **central processing unit** is the fastest part of the computer. The CPU consists of a number a very high-speed memory units called *registers*, containing the *instructions* sent to the hardware to do things like fetch, store, and operate on data.

ii) FPU: The **floating point** (or arithmetic) **unit** is a piece of hardware designed for the quick operation of floating-point arithmetic. On machines in the past, these chips were called *math co-processors*.

iii) Cache: A small, very fast bit of memory (sometimes called the *high-speed buffer*) that holds instructions, addresses, and data in their passage between the very fast CPU registers and the slower main RAM memory. The main memory is also called *dynamic* RAM (DRAM), while the cache is *static* RAM (SRAM).

iv) Cache and data lines: The data transferred to and from the cache or CPU are grouped into *cache lines* or *data lines*. The time it takes to bring data from memory into cache is called *latency*.

v) RAM: Random access memory or central memory is in the middle memory hierarchy. This is where your program resides while it is being processed. The contents of RAM is lost upon completion of the jobs (or the turning off of the machine). The RAM of your computer is analogous to the memory centers of your brain.

vi) ROM: Read only memory contains data that is hard-coded on the chip (typically, non-erasable). Information on this chip tells the machine its identity, senses the devices hooked to the motherboard (called *peripherals*), and tells the machine where to find the boot software. The ROM is analogous to

your DNA.

vii) Pages: Central memory is organized into **pages**, which are blocks of memory of fixed length. The operating system labels and organizes its memory pages much like we do with the pages of a book $\Longrightarrow$ they are numbered and kept track of in a *table of contents*.

viii) Hard disk: Finally, at the bottom of the memory hierarchy is the permanent storage on magnetic disks or optical devices. They are slow, but can store large amounts of data. The coding of the operating system, compilers, and any other documentation, whether in ASCII format or binary format are stored here.

ix) Backup devices: Since hard disks work very hard, they are the first thing that is likely to malfunction on a computer. Once the *head* crashes on a disk drive, the data stored there is typically lost forever. As such, it is good practice to store the contents of your hard drive (*i.e.*, hard disk) on a more permanent and safe medium: **USB Flash Drives**, **CD-ROMs**, and writable **DVDs**.

x) Virtual memory: Virtual memory permits your program to use more pages of memory than will physically fit into RAM at one time. Pages not currently in use are stored in slower memory (typically hard disks in a region called **swap space**) and brought into fast memory only when needed.

The amount of memory space needed for a program to run is limited by the machine's RAM plus the swap space size. In Unix, the amount of swap space available and total can be found with the "swap" (on some machines called "swapon") command.

2. The speed of the CPU is determined by the clock chip that is attached to it. The faster the clock (measured in MHz or GHz), the faster your CPU.

3. CPU clock speed doesn't tell the whole story in computer speed. The speed of the cache and the speed of the *bus* that talks to the hard disk (and other peripherals) are also important in "turn-around-time" in running a program.