Computational Foundations

Lecture Notes by Rangi Siebert

October 21, 2023

Contents

0.1 Preface

The three major topics:

Aspect	What	Why
Programming	Develop a solid understanding of C/C++ including both object orientation and embedded targets.	Most aerospace applications have requirements that are only fulfilled with C/C++.
Computer Science Foundations	To learn the basic notations like algorithms design, software engineering, computational complexity, data structures.	To safely navigate the complex and growing field of computational methods.
Digital electronics (basics)	To learn how binary logic and digital circuits lead to computers	For programming embedded systems, a certain level of low-level knowledge is helpful.

0.1.1 Motivation

Why C/C++:

- One of the **few active languages** with a long history (equally Fortran).
- The **language of choice for embedded** (Arduino, FPGAs, Flight Controller, Operating Systems, ...).
- The **language of choice for high performance** (MPI/HPC, Simulation, Real-Time, Vi-

sualization).

- It is **complete and non-simplified** (hard to learn, but good to know, all important concepts can be shown).

- It can be a **safe language** with sufficient education (here the history strikes back).

- It can be efficient to use as high quality libraries for **everything** are available.

Training the Next Generation of Scientists: New knowledge and skills will be needed to make efficient use of new system architectures and software. "Hybrid" disciplines such as computational science and data science and interdisciplinary teams may come to play an increasingly important role. Keeping abreast of a rapidly evolving suite of relevant technologies is challenging for many computer science programs, especially those with limited partnerships with the private sector. Most domain scientists rely on traditional software tools and languages and may not have ready access to knowledge or expertise about new approaches.

→ Domain Scientists need to get somewhat independent from traditional domain software tools.

...

1 Computers

Personal computers have always been sold as a computing appliance including both software and hardware (OEM - "Original Equipment Manufacturer"). Even today, most computers are shipped with:

- MS Windows

- Apple Mac OS

- Linux

1.1 MS DOS

Central Idea: A small operating system to manage disks (floppy disks, hard drives) and to run programs in relation to files. From these early days, a built-in feature has survived to today: **The Command Line (CMD)**

1.1.1 The CMD

- **Some Elements**

<drive letter>:	to change the drive
CD	to change the directory
DIR	to list a directory, consider switches /P and /S which change the behavior
MD	to create a directory
RD	to remove a directory (only if empty)
TREE	shows all files below the current working directory
ATTRIB	show and modify attributes like write protection on files
COPY	is used to copy files
DEL	deletes files (synonymous with ERASE)
EDIT	provides a simple editor (EDLIN before MS DOS 6.0)
FIND	searches for a string in a file
MORE	pages a file to the screen
MOVE	moves a file to a different location
PRINT	is used to print a file
REPLACE	works like copy but replaces the file in the target
TYPE	outputs the whole content of the given file
XCOPY	extends COPY to be able to copy whole directories and trees
CLS	clears the screen
DATE	shows and modifies the date
TIME	shows and modifies the time
ECHO	is used to control whether commands are shown or not (mainly in batch files)

FDISK	is used to set up hard disks (partitions, etc.)
FORMAT	organizes a file system on floppy disks or hard disk partitions
HELP	shows help for a dos command (use it in the tutorial!)
SET	shows configuration information and environment variables and modifies them
VER	shows the version of DOS in use

1.1.2 OS / Program Interaction

In classical MS DOS, a program was supposed to replace the whole operating system and take full control over the computer (e.g., immediate mode, all memory directly accessible)

Information from the operating system was typically transferred through certain state including:

- Environment variables.
- The current working directory CWD (progress working directory on linux, pwd).
- A sequence of arguments.

1.1.3 File Patterns

A special helpful situation was that file names could be specified incomplete using special characters:

- * which could stand for zero or more arbitrary characters.
- ? which could stand for a single arbitrary character.

Together with the fact, that all files in DOS (and Windows today) have a file extension specifying the type, it became very easy to work on specific files. For example,

```
MD texts
COPY *.txt texts
```

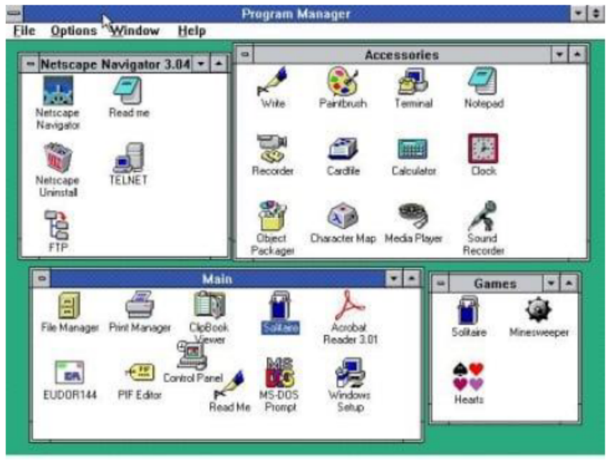
would first create a directory texts below the CWD and then copy all text files to this directory

1.1.4 Automation Using the Command Line

Many times, computers are used for routine tasks. And in many cases, you get no support from your IT. It is the nice to know that a stock windows provides enough to automate tasks and that a Linux is even more equipped with onboard automation tools. (.sh and .bat files)

1.2 Microsoft Windows

Windows has emerged over a few development steps, first being a graphical file manager, but the first relevant version of Windows is maybe Windows 3.1 (and Windows for Workgroups 3.11) together with the professional versions of Windows NT.



1.2.1 Key Idea

The key idea of Windows is that interaction with a computer can have multiple parallel scopes represented by Windows out of which exactly one is active. This active window is the one receiving events from the user interface elements (keyboard & mouse events). The event-driven architecture manifests itself in the structure of a Windows application which consists of a single Message Queue to which messages are delivered.

1.3 Linux (and other UNIXes)

1.3.1 Unix Philosophy

A famous formulation of the Unix philosophy is due to Douglas Mcilroy:

Write programs such that they ...

... do only one thing and they to it well

... can work together.

... work on textual streams as this is the universal interface.

KISS principle: Keep it simple stupid.

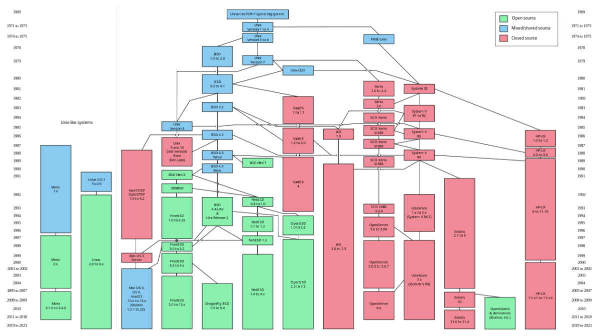
Early software engineering:

- Small is beautiful.
- Make each program do one thing well.
- Build a prototype as soon as possible.
- Choose portability over efficiency.
- Store data in flat text files.
- Use software leverage to your advantage.
- Use shell scripts to increase leverage and portability.
- Avoid captive user interfaces.
- Make every program a filter.

Other:

- Everything is a file.
- If it is a file, it should be a text file.

1.3.2 Unix History



1.3.3 POSIX

Maybe the most influential standard for programming and operating systems ever developed

- Term Definitions
- Application Interface (API functions, including C headers)
- Mandatory Utilities

Coreutils is a set of programs that should be available on all systems (gnu.org/software/coreutils/manual/coreutils.pdf). Learning a selection of them will be helpful over time, but not as part of this lecture.

1.4 The Web

The Web as an Application Platform: Using **remote applications** over a web browser.

- Application runs in a "backend".
- Frontend runs in a browser on various devices.
- Users need to authenticate (username/password, OpenID, etc.).
- Computational Power shifts to the backend.
- Cloud Computing provides elastic performance to such applications.
- Big Data is easier

Explore: Google Colab, Google Earth Engine, Microsoft Planetary Computer, Jupyter Notebooks (tutorial).

1.5 Relevant Projects and Languages

You need to learn:

- **Formatted programming** in HTML and JavaScript
- **Backend programming** in PHP, Python and Jekyll (e.g., Github Pages)

And much more beyond this lecture...

1.5.1 Typical Properties

- No software on client.
- User-level authentication.
- Medium performance applications (some faster, some slower than on-hos computing).

Programming in two components:

- The frontend is mostly written in HTML / JavaScript for modern browsers.
- The backend can be written in any programming language

2 Imperative Programming

2.1 Algorithms

Everyday Algorithms: Cooking a meal - A simple recipe; Finding the right heater setting; Sort playing cards.

Numbers - Integers: Written Addition; Written Subtraction; Written Multiplication; Division.

Numbers - Fractions: Detection Prime Numbers; Prime Decomposition; Greatest common divisor; Comparing fractions; Reducing fractions; Adding fractions; Multiplying fractions.

Games: Tic Tac Toe; Sudoku; Chess - queen end game; Chess - Two knight does not win (unless ...); 17+4; Minesweeper (or Kaboom).

Higher Mathematics - Linear Algebra: Finding Eigenvalues; Inverting a matrix; Finding the square root of a number.

Higher Mathematics - Analysis: Finding the root of a real-valued fraction; Computing the square root of a number; Computing the integral of a function.

2.2 The Theory of Algorithms

Free-form Algorithm Introduction:

Algorithm 1. *Ond draws a circle with the same radius r around A and B. If the radius is large enough (greater than half the distance), we are left with two intersection points of the circles which together define a line. This line is the bisector and the intersection of this respective line with the original line is the middle polint p.*

Assume we have a device (or person, or computer, or ...) that enables us to

- draw a straight line between two points in space,
- observe intersections of pairs of lines or pairs of circles resolving them to points,
- draw a circle around any of the involved points such that the circle intersects an already existing point.