Programming

Lesson 1. Course overview. Introduction to programming

Course overview

1	Algorithms, programming, programming languages. Algorithmizing problems. Methods of description for algorithms. Algorithms with linear, branched and cyclic structure.
2	Basic elements of the C programming language. Structure and syntax of C programs. Function main(). Data types and variables.
3	Data input and output. scanf () and printf () functions. C++ input and output

Course overview

4	**C expressions and operators. Operands and operators. C keywords and library math functions. Simple and Composite (Structured) expressions. Priority of operations.**
5	Selections. If and switch. Variations. Nested if. Implementation. Relational operations Logical experssions and functions.

Course overview

5	Loops. Counted controlled and condition-controlled loops. Pre and post conditions. Nested loops. Applications.
6	Arrays. Arrays declaration and initialization. Assigning values to array’s elements. Displaying array content. Algorithms for one- and two-dimensional arrays processing. Sorting arrays.

Course overview

7	Pointers. Operations & and *. Operations with pointers. Arrays and pointers. Operations with pointers on arrays. Arrays of pointers. Pointer to the array. Equivalent notations.
8	User defined Functions. Function body, definition and call. Values return, Function parameters. Data exchange between functions. Local and global variables. Time of existence and visibility. Memory classes.

Course overview

9	Methods of memory allocation in C. Standard allocating, reallocating, and free dynamic memory. Dynamic assignment of one-dimensional and two-dimensional arrays
10	Character processing. Read and display characters. Standard character processing. String processing. Array of chars and string in C. Operations on strings and arrays of char. Input / output data. Functions to process strings.

Course overview

11	File Processing Pointer to file. Opening and closing a file. Standard functions for file processing
12	Recursion. Introduction. Recursive functions. When to use recursion? Direct and indirect recursion. Recursive problem solving.

Course overview

13	User data Types. Structures. Access to structure elements (fields). The typedef statement. Unions. Bit maps. Enumerative types. Arrays of structures processing.
14	C-preprocessor directives. Macro Definition and pseudo function. Conditional compilation. Arguments of the main () function.

Seminars

Course overview

1 **Project 1.** Hello World
1.1 Data input / output
1.2 User scanf and user printf
1.3 Formatted output
1.4 Draw in pseudographics

2 **Project 2** Simple algorithms
2.1 Babylon algorithm for square root
2.2 Euclid's algorithm for GCD
2.3 Prime numbers
2.4 Perfect numbers (the sum of divisors)

Course overview

3	**Project 3.** Long numbers operations (Arrays) 3.1 Big numbers sum 3.2 Big numbers difference 3.3 Big numbers multiplication
4	Project 4. Slow sorting algorithms 4.1 Bubblesort 4.2 Insertion sort 4.3 Selection sort 4.4 Time estimation and counting

Course overview

5	Project 5 Combining arrays 5.1 3x3 o – X game 5.2 infinite Ran Zee game 5.3 Snake
6	Project 6. Bit by bit 6.1 binary arithmetic + power 2 fast calculation 6.2 binary logics 6.3 Logical function evaluation

Course overview

Project 7 Characters processing 1:
7.1 Development of string operation and conversion functions
7.2 Proto Zuma game development

Project 8 Characters processing 2
8.1 reading and editing text from file
8.2 formatting text files

Team projects

Course overview

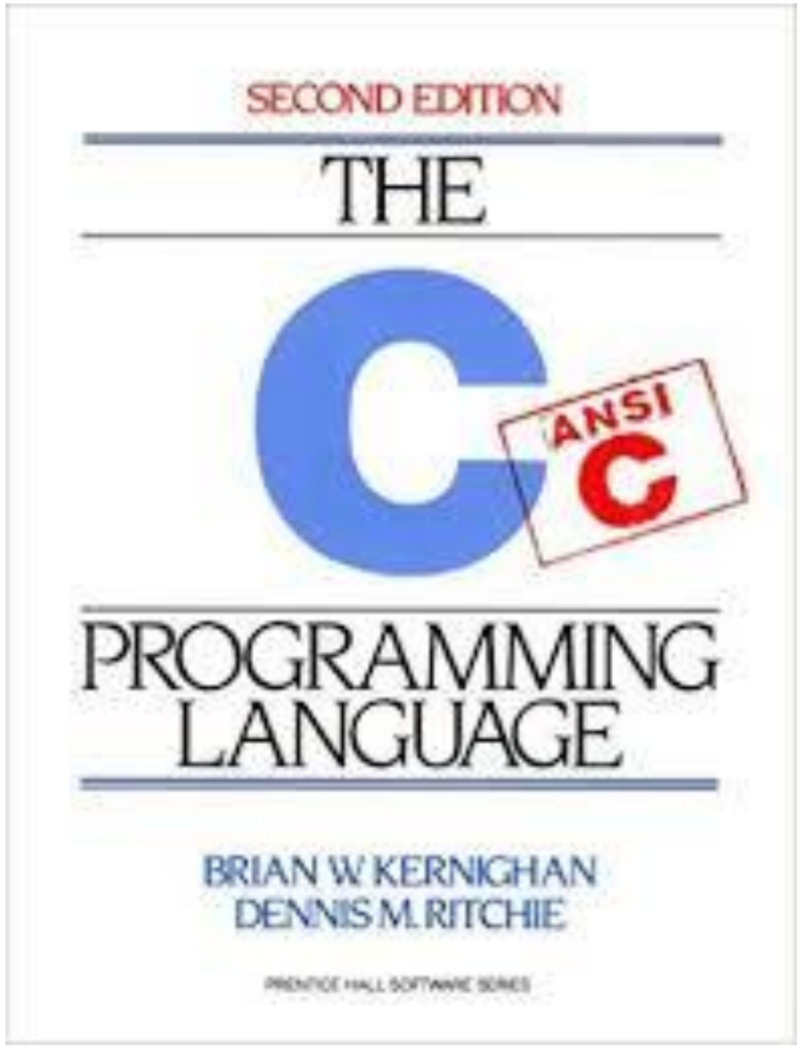
Project 1. Groups of 5. Random numbers generators (October 10)

Project 2. Groups of 5. The magic of π (November 10)

Project 3. Groups of 5. The "Life" game (December 10)

Books

Basic 6

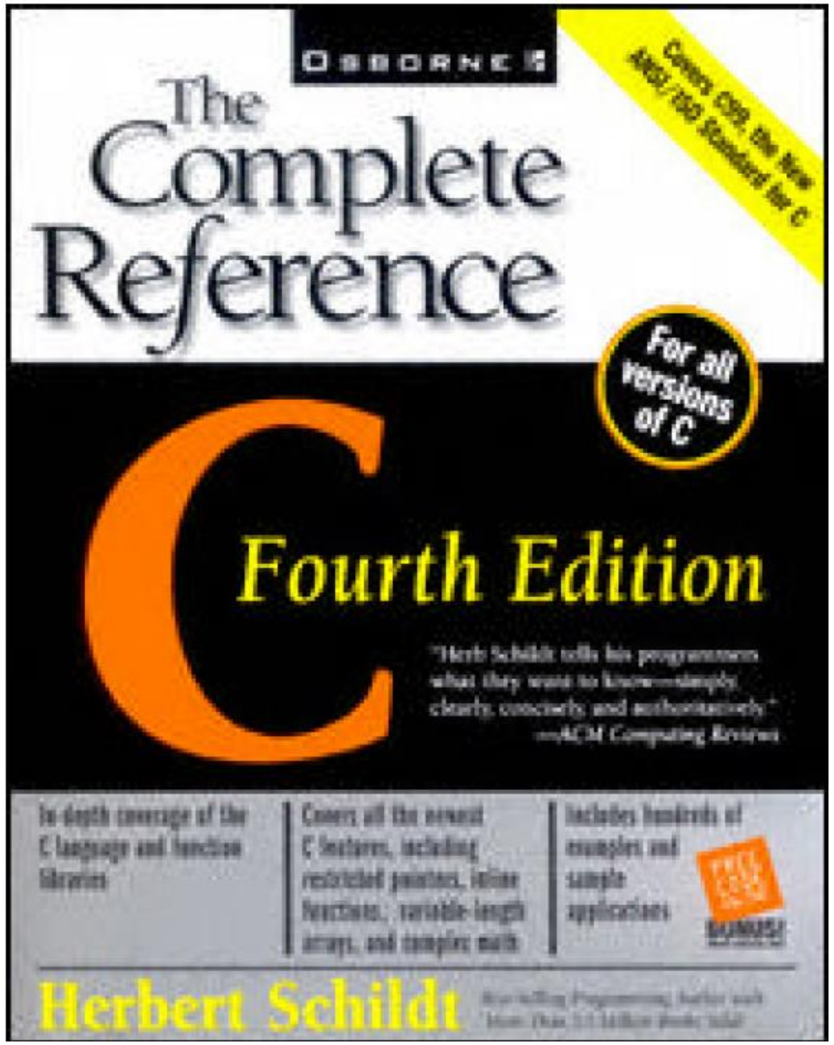

SECOND EDITION

THE C PROGRAMMING LANGUAGE

ANSI C

BRIAN W. KERNIGHAN
DENNIS M. RITCHIE

PRENTICE HALL SOFTWARE SERIES


OSBORNE

Covers C99, the New ANSI/ISO Standard for C

The Complete Reference

For all versions of C

C Fourth Edition

"Herb Schildt tells his programmers what they want to know—simply, clearly, concisely, and authoritatively."
—ACM Computing Reviews

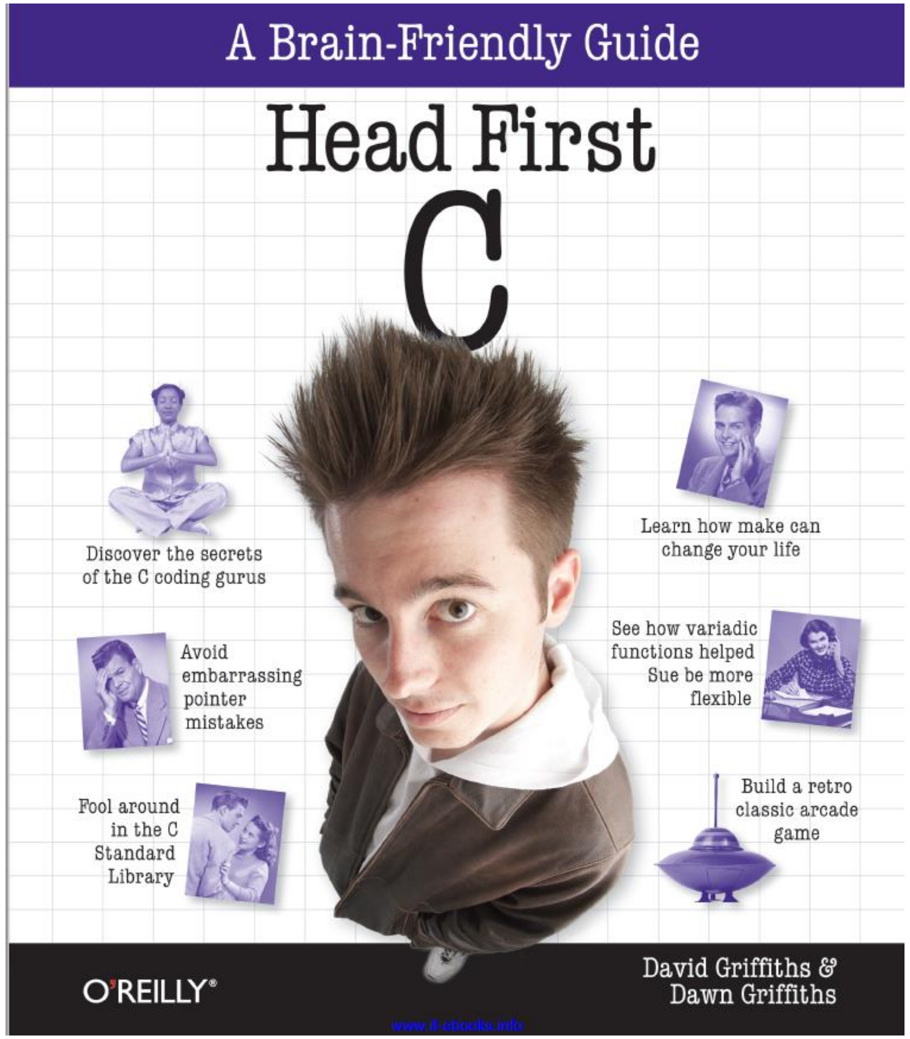
In-depth coverage of the C language and function libraries

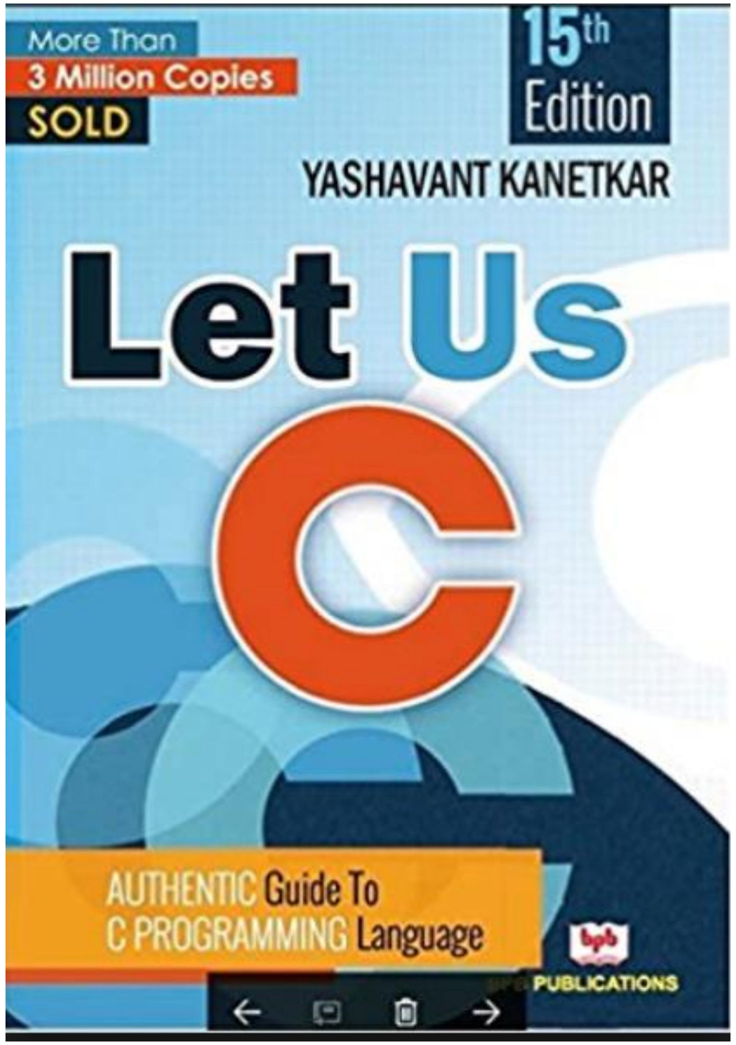
Covers all the newest C features, including restricted pointers, inline functions, variable-length arrays, and complex math

Includes hundreds of examples and sample applications

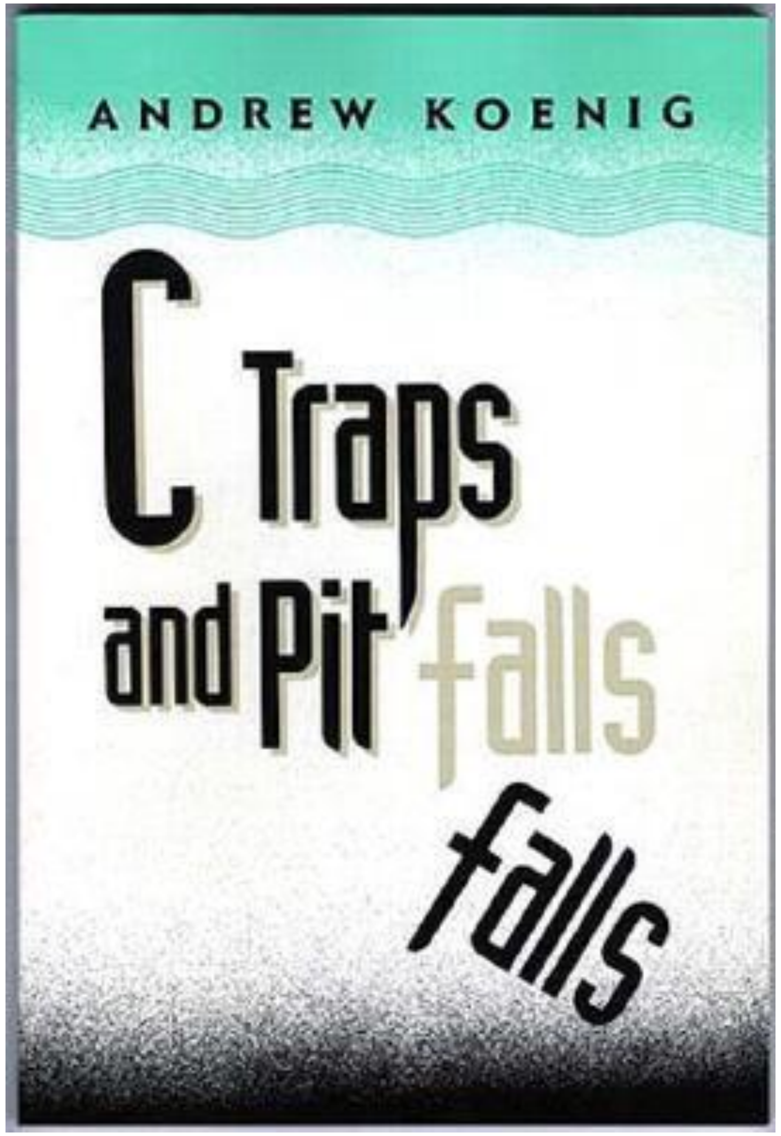
Herbert Schildt Best-Selling Programming Author with More Than 3.5 Million Books Sold!

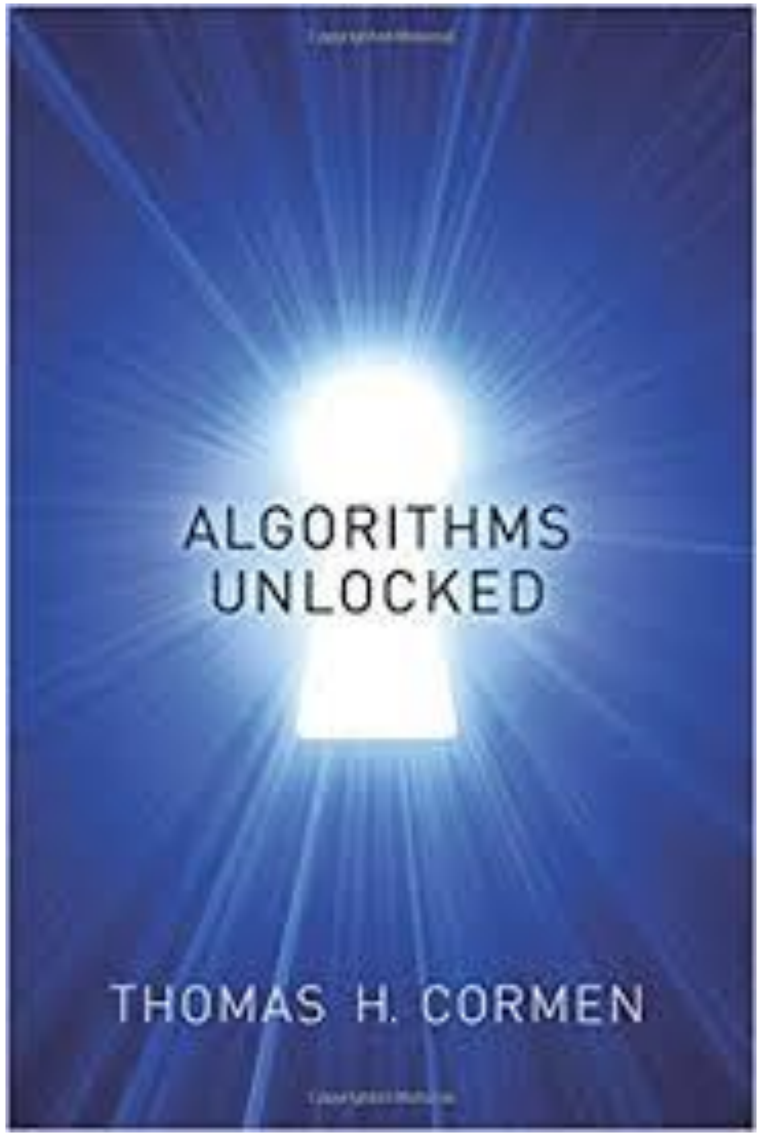
Basic 5


A Brain-Friendly Guide
Head First
C
Discover the secrets of the C coding gurus
Avoid embarrassing pointer mistakes
Fool around in the C Standard Library
Learn how make can change your life
See how variadic functions helped Sue be more flexible
Build a retro classic arcade game
O'REILLY
David Griffiths & Dawn Griffiths

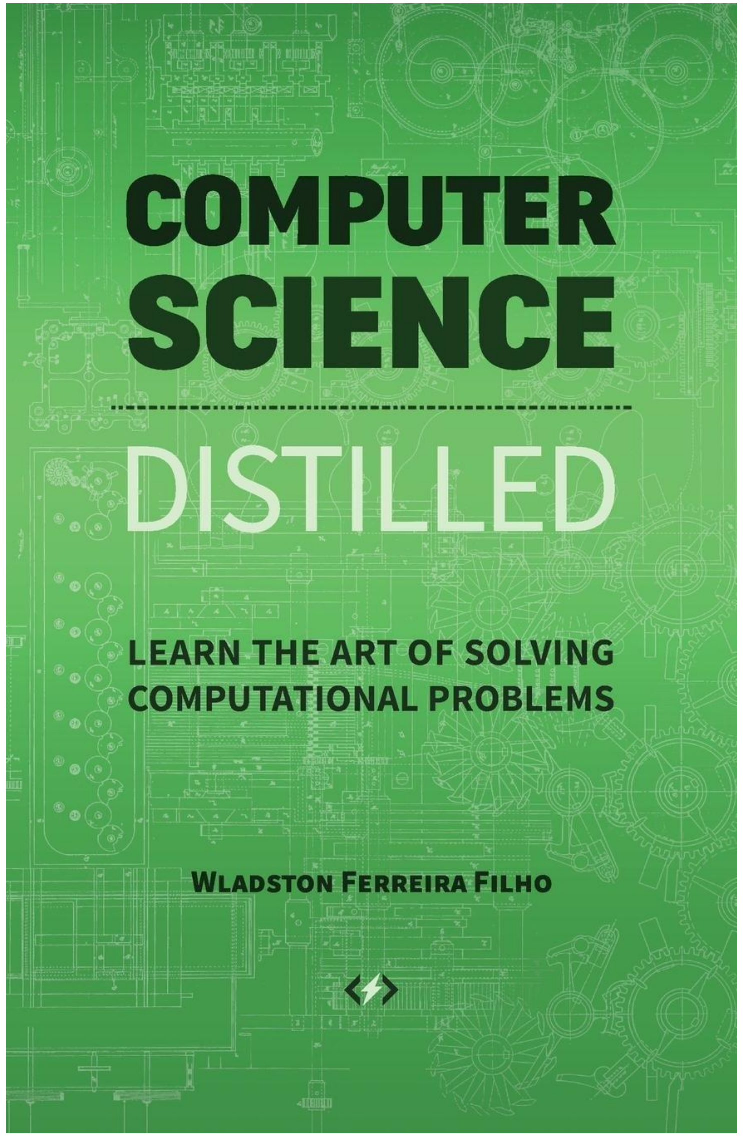

More Than
3 Million Copies
SOLD
15th Edition
YASHAVANT KANETKAR
Let Us
C
AUTHENTIC Guide To
C PROGRAMMING Language
PUBLICATIONS

Basic 6


ANDREW KOENIG
C Traps and Pitfalls
falls

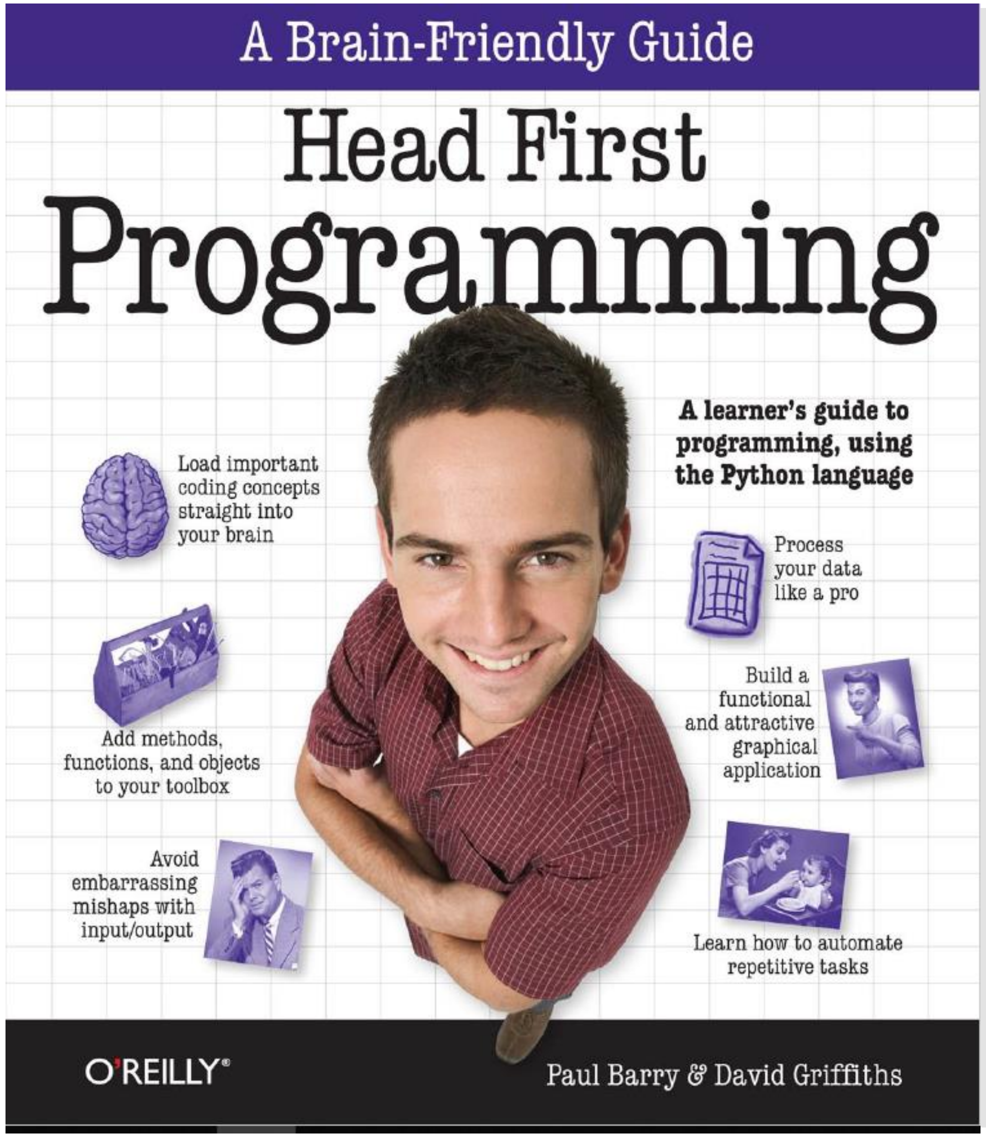

ALGORITHMS UNLOCKED
THOMAS H. CORMEN

More 2


COMPUTER SCIENCE DISTILLED

LEARN THE ART OF SOLVING COMPUTATIONAL PROBLEMS

WLADSTON FERREIRA FILHO


A Brain-Friendly Guide

Head First Programming

A learner's guide to programming, using the Python language

Load important coding concepts straight into your brain

Process your data like a pro

Add methods, functions, and objects to your toolbox

Build a functional and attractive graphical application

Avoid embarrassing mishaps with input/output

Learn how to automate repetitive tasks

O'REILLY®

Paul Barry & David Griffiths

What is Programming?

Lesson 1. Part 2.

Programming	- The planning, scheduling of a task or an event.
Computer	- often attrib (1646): one that computes; specif: a programmable electronic device that can store, retrieve, and process data
Computer Programming	- The process of planning a sequence of instructions for a computer to follow.
Computer Program	- A sequence of instructions outlining steps to be performed by a computer.

Our definition of programming leaves a lot unsaid- In order to write a sequence of instructions for a computer to follow, we must go through a certain process.

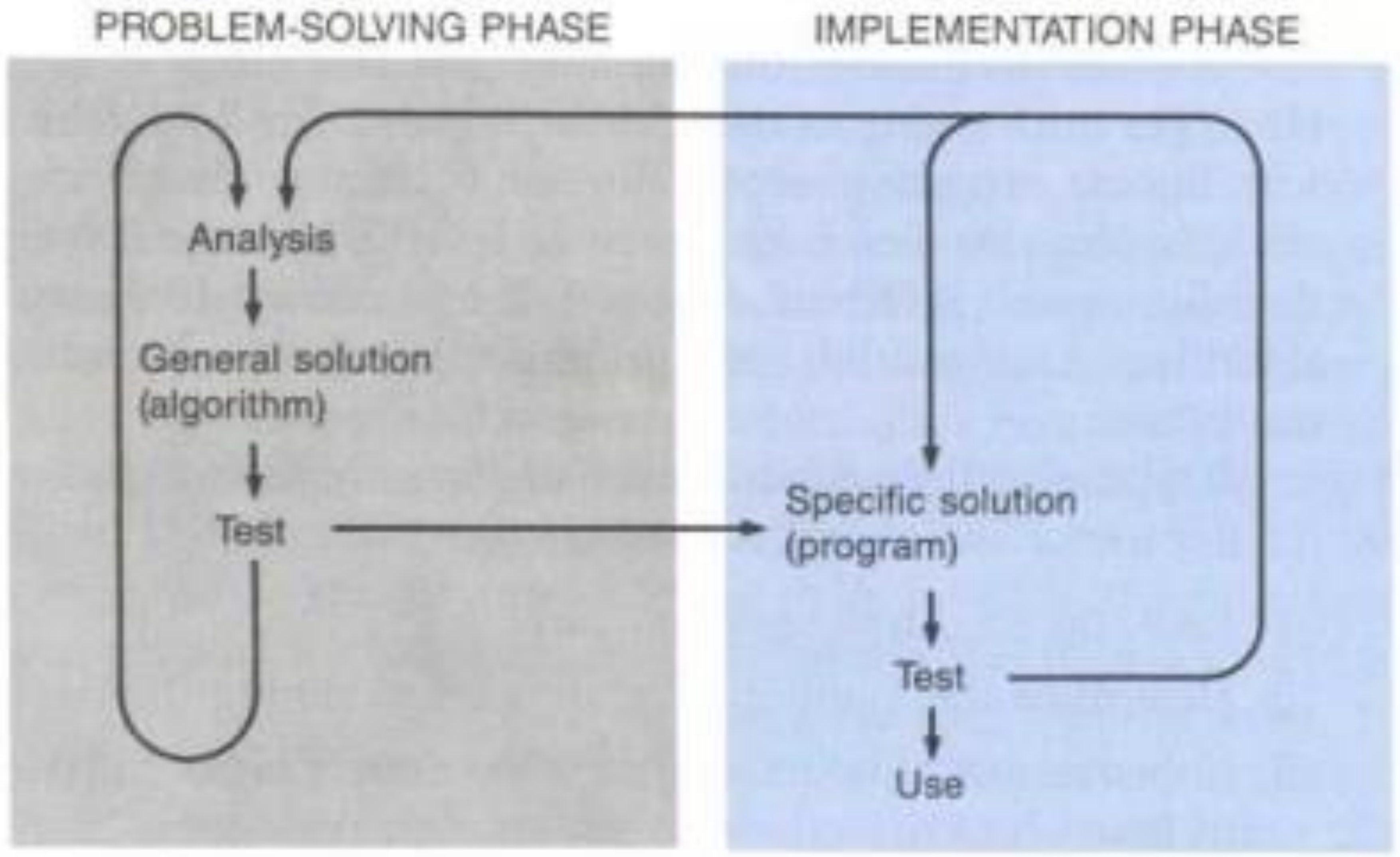
This process is composed of a *problem-solving phase* and an *implementation phase* .

Problem-Solving Phase

- **Analysis** — Understand (define) the problem.
- **General Solution (Algorithm)** — Develop a logical sequence of steps to be used to solve the problem.
- **Test** — Follow the exact steps as outlined to see if the solution truly solves the problem.

Implementation Phase

- **Specific Solution (Program)** — Translate the algorithm into a programming language (code).
- **Test** — Have the computer follow the instructions. Check the results and make corrections until the answers are correct.
- **Use** — Use the program.


PROBLEM-SOLVING PHASE
IMPLEMENTATION PHASE
Analysis
General solution (algorithm)
Test
Specific solution (program)
Test
Use

Meditations…

The computer, alas, is not intelligent. It cannot analyze a problem and come up with a solution. **The programmer must arrive at the solution and communicate it to the computer**.

The problem solving is done by the programmer — not the computer.

So, you may ask, what's the advantage of using a computer if it can't solve any problems?

Meditations…

The advantage is that once we have a solution for a problem and have prepared a version of it for the computer, the computer can rapidly repeat the solution again and again and again.

Thus the computer may be used to free people from tasks that are repetitive and boring or that require great speed or consistency.

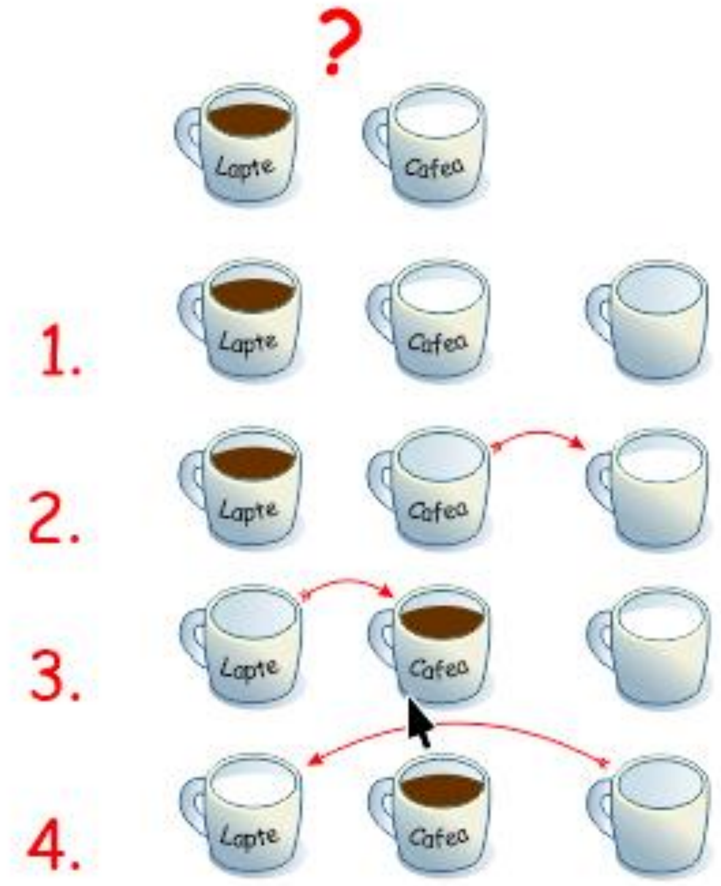
Algorithms

The programmer begins the programming process by analyzing the problem and developing a general solution called an *algorithm* .

Understanding, analysis of a problem and algorithm developing are the heart of the programming process.

Algorithms

Algorithm	- A step-by-step procedure for solving a problem in a finite amount of time



Start the car

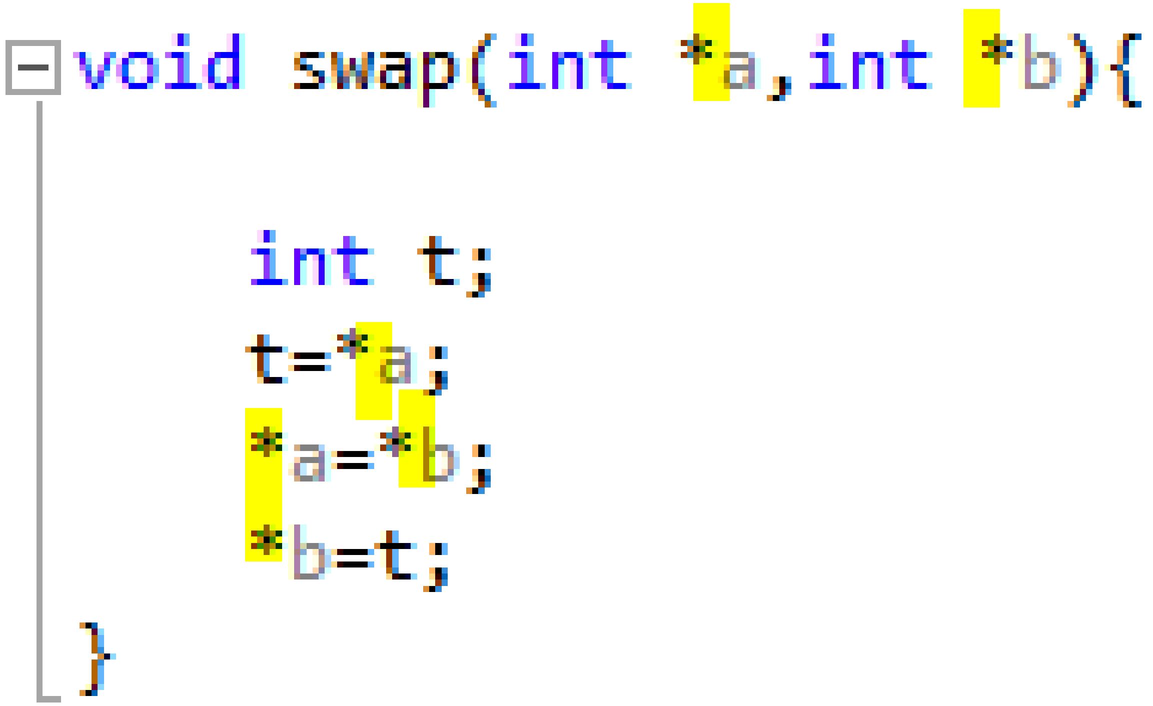
1. Insert the key.
2. Make sure the transmission is in Park (or Neutral).
3. Depress the gas pedal.
4. Turn the key to the start position.
5. If the engine starts within six seconds, release the key to the ignition position.
6. If the engine doesn't start in six seconds, wait ten seconds and/repeat steps 3 through 6 (but not more than five times).
7. If the car doesn't start, call the garage.

Go to next page (reffers to a book)

1. Lift hand.
2. Move hand to right side of book.
3. Grasp cover of top page.
4. Move hand from right to left until page is positioned so that what is on the other side can be read.
5. Let go of page.

Algorithms

Computer Algorithm	- In computer systems, an algorithm is basically an instance of logic written in software by software developers, to be effective for the intended "target" computer(s) to produce output from given (perhaps null) input.



```
void swap(int *a,int *b){

    int t;
    t=*a;
    *a=*b;
    *b=t;
}
```

Programming Languages

After developing a general solution, the programmer "walks through" the algorithm by performing each step mentally or manually. If this testing of the algorithm doesn't produce satisfactory answers, the programmer repeats the problem-solving process, analyzing the problem again and coming up with another algorithm. Often the second algorithm is just a variation of the original one.

When the programmer is satisfied with the algorithm, it is translated into a *programming language*.

We use a programming language, rather than English, because a programming language restricts us to writing only instructions that the computer can carry out.

Programming language	- A set of rules, symbols and special words, used to construct a program

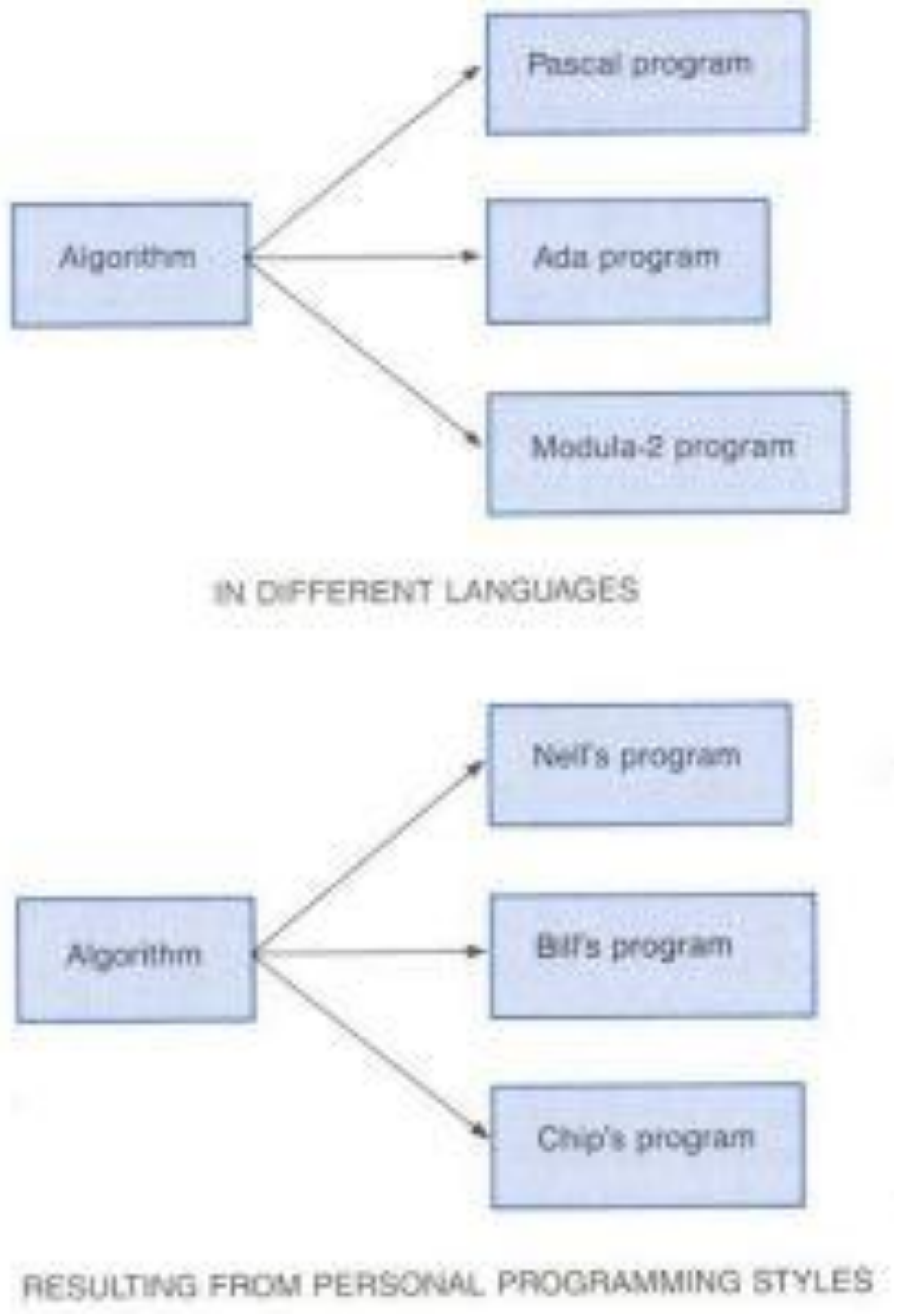
Implementation

Translating an algorithm into a programming language is called ***coding*** the algorithm.

The resulting program is ***tested by running*** (*executing*) it on the computer. If the program fails to produce the desired results, the programmer must determine what is wrong and modify the algorithm and program as needed.

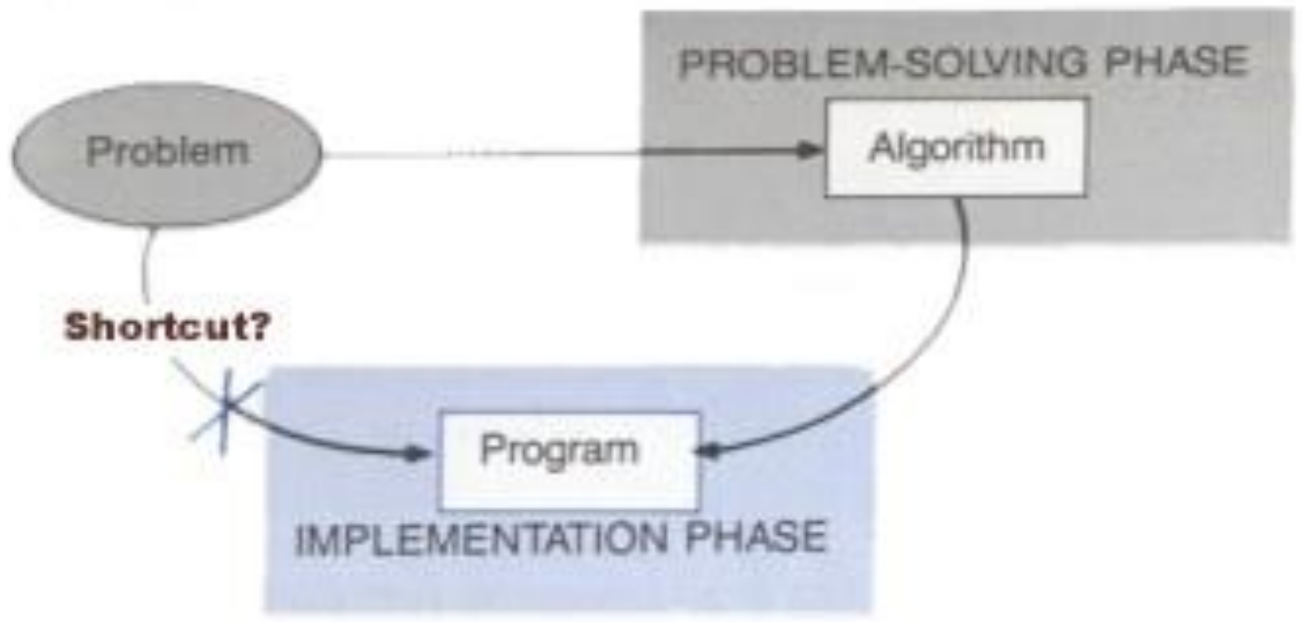
The combination of coding and testing an algorithm is often referred to as ***implementing*** an algorithm.

Differences in implementation



The Basic Instinct –

The Basic Mistake



Programming languages

Mashine language

Remember that in the digital computer all data, whether it is alphabetic or numeric data or instructions, is stored and used in binary codes of **1**'s and **0**'s. When computers were first developed, the only programming language available was the primitive instruction set of each machine, known as ***machine language*** (or machine code).

Even though most computers perform (he same kinds of operations, different designers have chosen different sets of binary codes of 1 's and 0's to stand for each instruction. **So the machine code for one computer is not the same as for another.**

Machine language	- The language used directly by the computer and composed of binary-coded instructions.

Programming languages

Assembly language

When programmers used machine language for programming, they had to enter the binary codes for the various operations of the computer, a tedious and error prone process. The resulting programs were difficult to read and modify, so ***assembly languages*** were developed to make the programmer's job easier.

Assembly Language	- A low-level programming Language in which a mnemonic is used to represent each of the machine language instructions for a particular computer.

An instruction in assembly language is in an easy-to-remember form called a mnemonic (pronounced "ni-mon'ik"). Typical instructions for addition and subtraction might look like this:

Assembly Language	Machine Language
ADD	100101
SUB	010011

The only problem with assembly language was that intinctions written in it could not be directly executed by a computer. So a program called the **assembler** was written to translate the assembly language instructions into machine language instructions (machine code).

Assembler	- A program that translates an assembly language program into machine code.

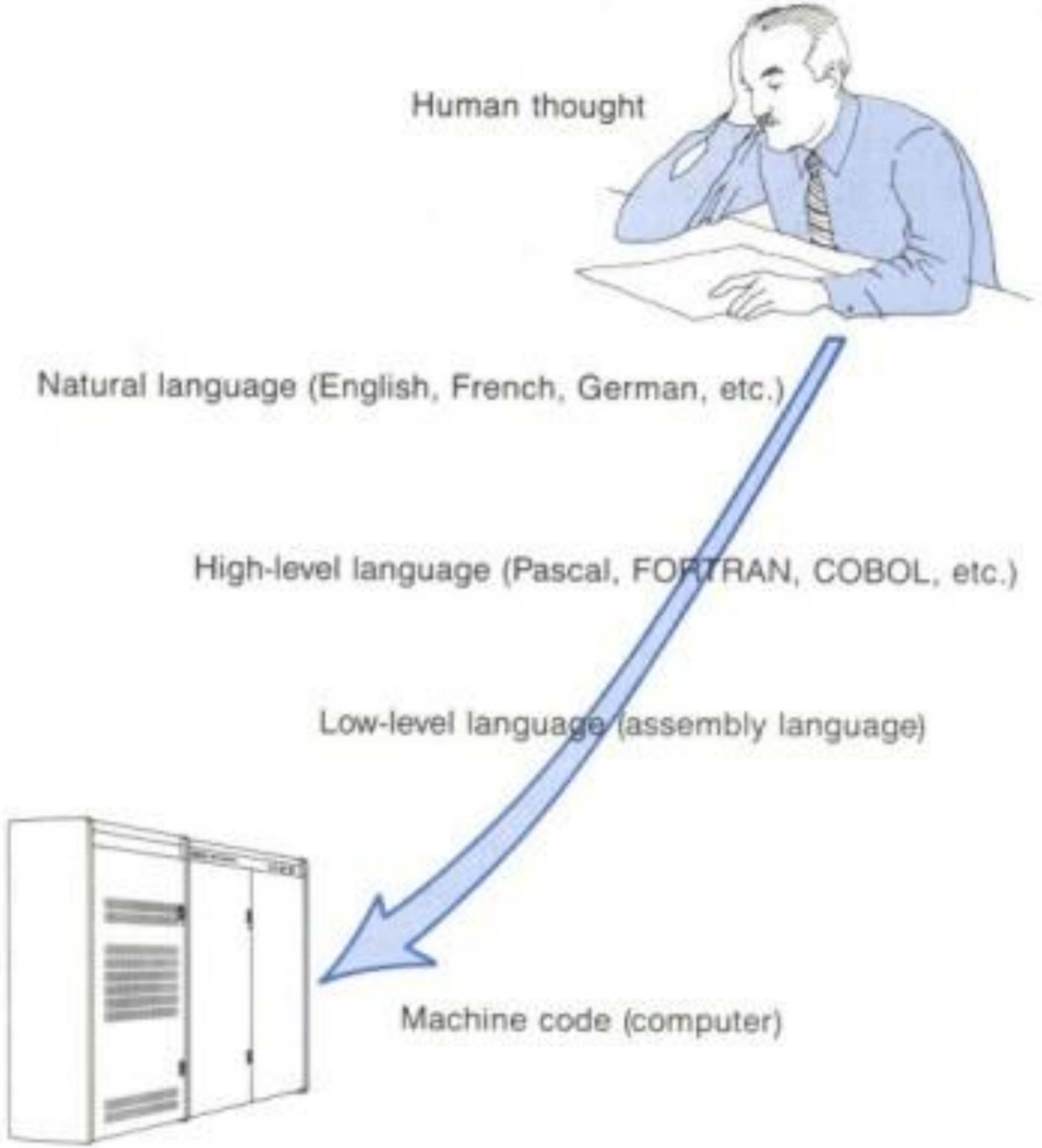
High level programming languages

Development of the assembler was a step in the right direction, but a programmer was still forced to think in terms of individual machine instructions.

Eventually high-level programming languages were developed that were closer to natural languages such as English and thus less limiting than machine code.

Programs in these high-level languages (such as C, COBOL, JAVA, Python etc.) must be translated into machine language instructions by a translator program called the ***compiler***, or, in some cases - ***interpreter*** .

Compiler	- A program that translates a high-level language into machine code.


Human thought
Natural language (English, French, German, etc.)
High-level language (Pascal, FORTRAN, COBOL, etc.)
Low-level language (assembly language)
Machine code (computer)

If we write a program in a high-level language, we can run it on any computer that has the appropriate compiler. This is because most high-level languages are *standardized*, meaning that an official description of the language exists.

A program in a high-level language is called a *source program*. When the source program is compiled by running the compiler with the program as data, a machine language program called an *object program* results. The object program can be run directly on the computer.

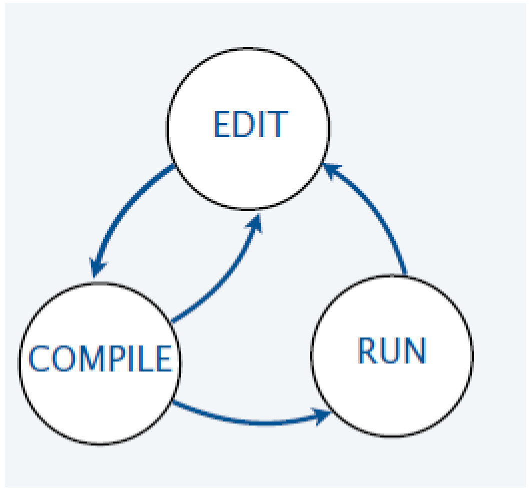
Notice that compilation/execution is a two-step process. During the compilation phase of the process, the computer is running the compiler program.

To the compiler program, a source program is just input data. The compiler program outputs a machine language translation of the source program. In the execution phase of the process, the machine language version of the program (object code) replaces the compiler program in the computer's memory. The computer then runs the object program, doing whatever the program instructs it to do.

Source Program	- A program written in a high-level programming language.
Object Program	- The machine language version of a program that results when a compiler translates a source program into binary codes for a particular computer.

Program development in C / C++

1. Write your code
   - Create it by typing on your computer's keyboard.
   - Result: a text file such as HelloWorld.cpp.
2. Compile your code
   - Use the C / C++ compiler
   - Result: a C bytecode file such as HelloWorld.o
   - Error? Go back to 1. to fix and recompile.
3. RUN your program
   - • Use the exe file.
   - • Result: your program’s output.
   - • Wrong answer? Go back to 1. to fix, recompile, and run.



Any creative process involves cyclic refinement / development.

My first C program

Tools

C IDEs and compilers

a) Dev C / C++ (local installation)
b) CodeBlocks (local installation)
c) Visual C (Microsoft) (local installation)
d) OnlineGDB (web)
e) CSAcademy.com (web)

Etc.

IDE - Integrated Development Environment

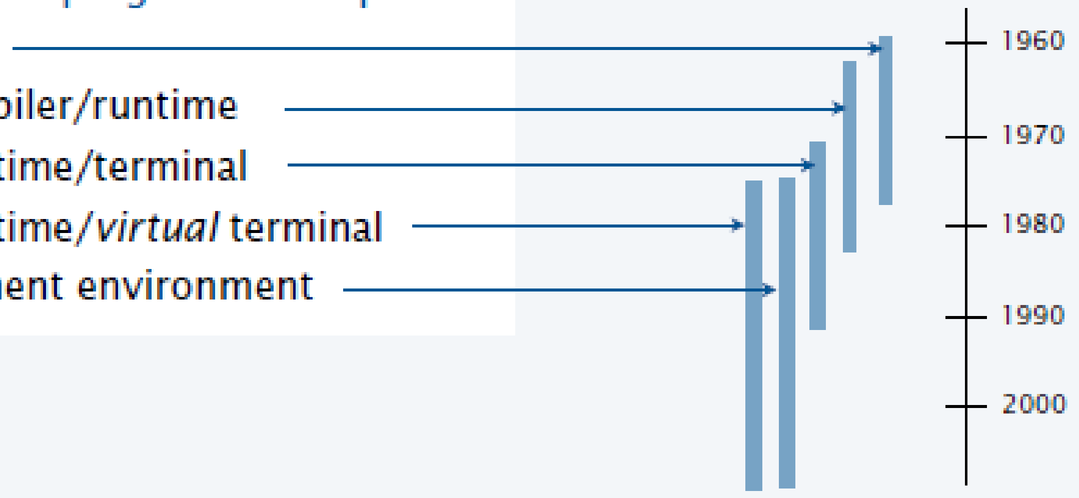
Historical background

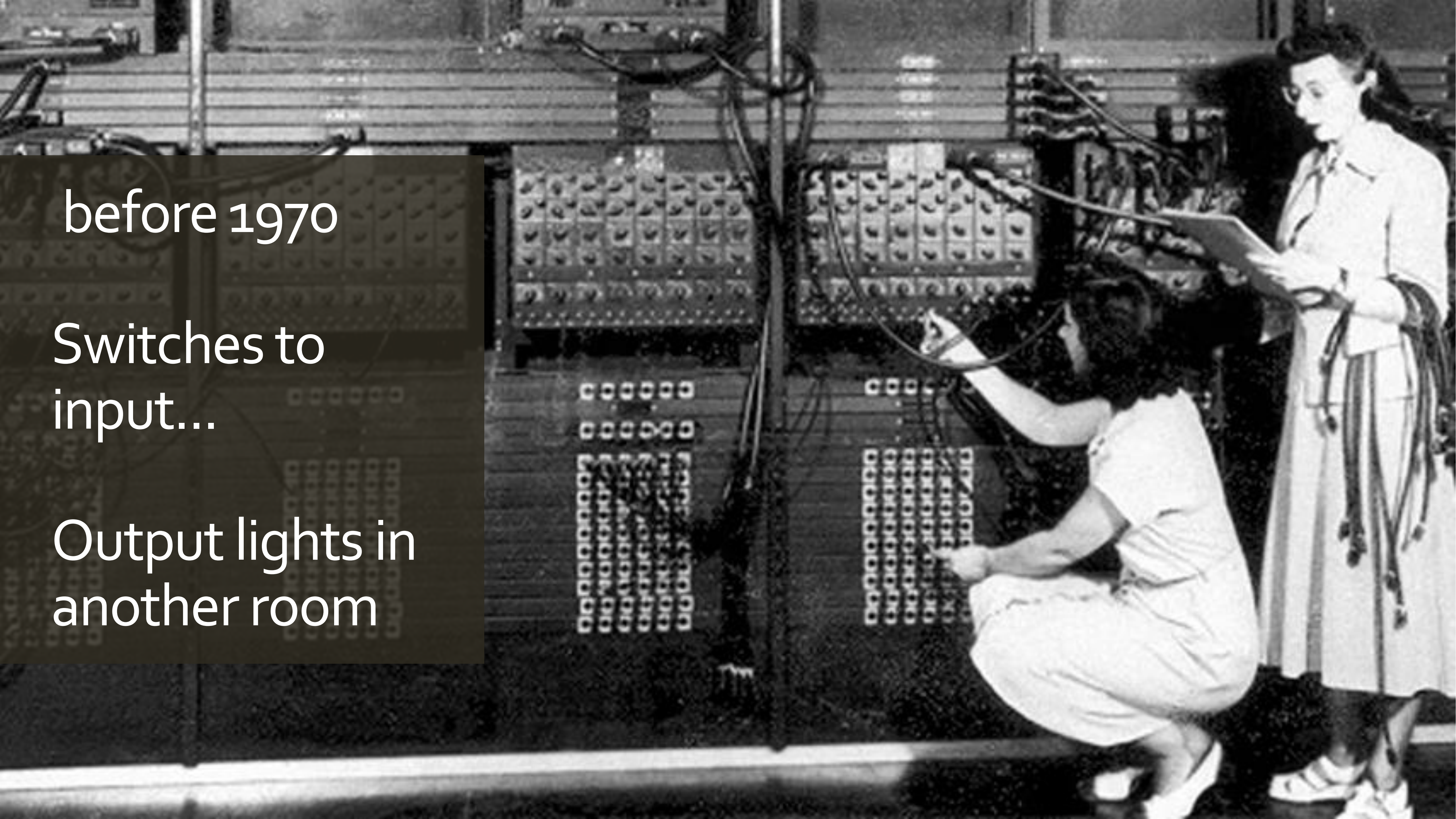
Historical context is important in computer science.

- We regularly use old software.
- We regularly emulate old hardware.
- We depend upon old concepts and designs.

Widely-used methods for program development

- switches and lights
- punched cards/compiler/runtime
- editor/compiler/runtime/terminal
- editor/compiler/runtime/*virtual* terminal
- integrated development environment





before 1970

Switches to input…

Output lights in another room

Card Reader Service for 80-Column IBM Punch Cards http://PunchCardReader.com

mid 1970

Punched cards to input...

Line printer to output

late 1970

first terminals (keyboards) to input…

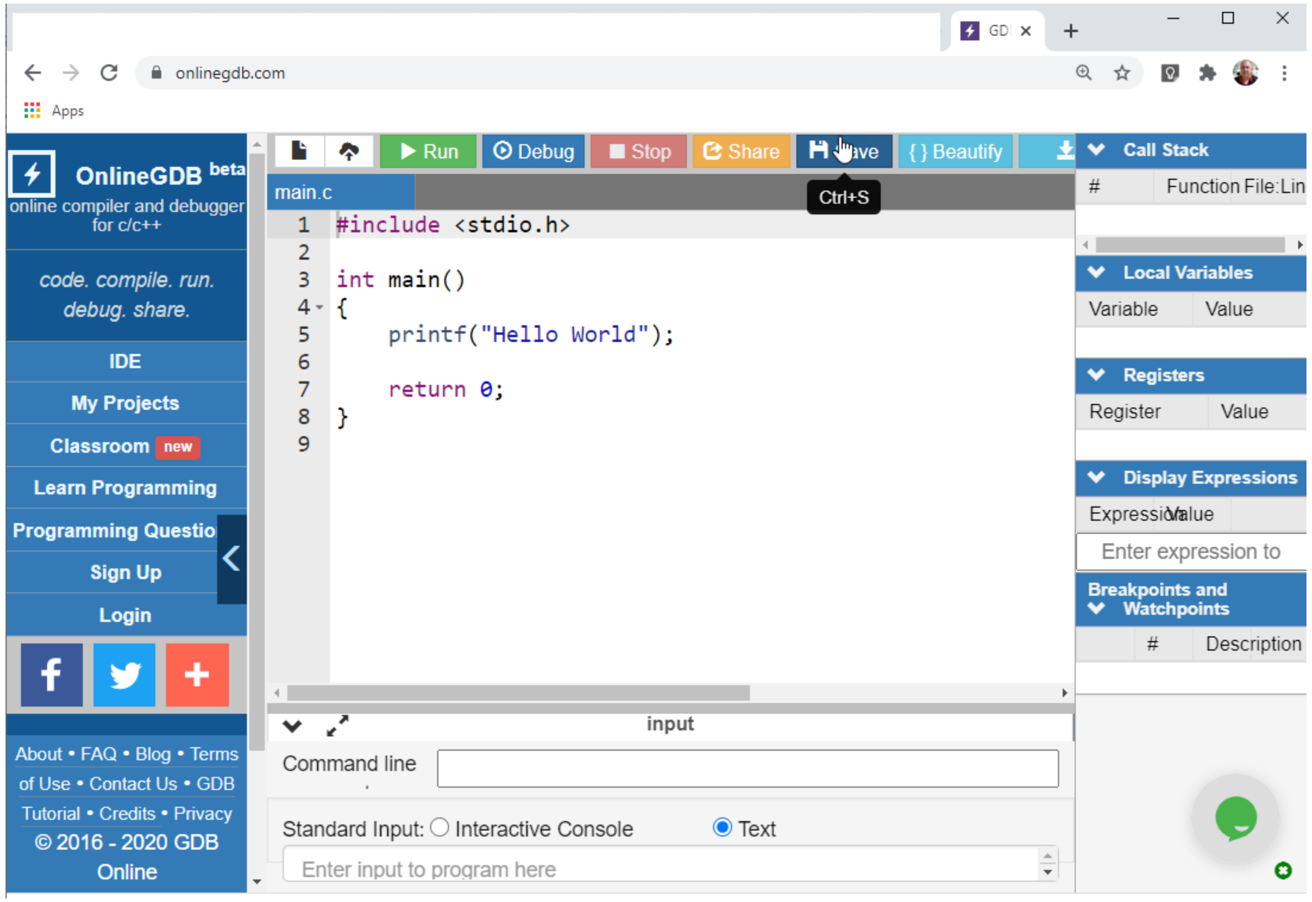
and output (monitor)

1980 to present…

program development with personal computer

1980s to present day: Use multiple *virtual terminals* to interact with computer.

- Edit your program using any text editor in a virtual terminal or IDE.
- Compile it in another virtual terminal or in IDE.
- Run it by typing HelloWorld or **Run** button



```
#include <stdio.h>

int main()
{
    printf("Hello World");

    return 0;
}
```

Personal computers

2 x FDD of 720 KB each!

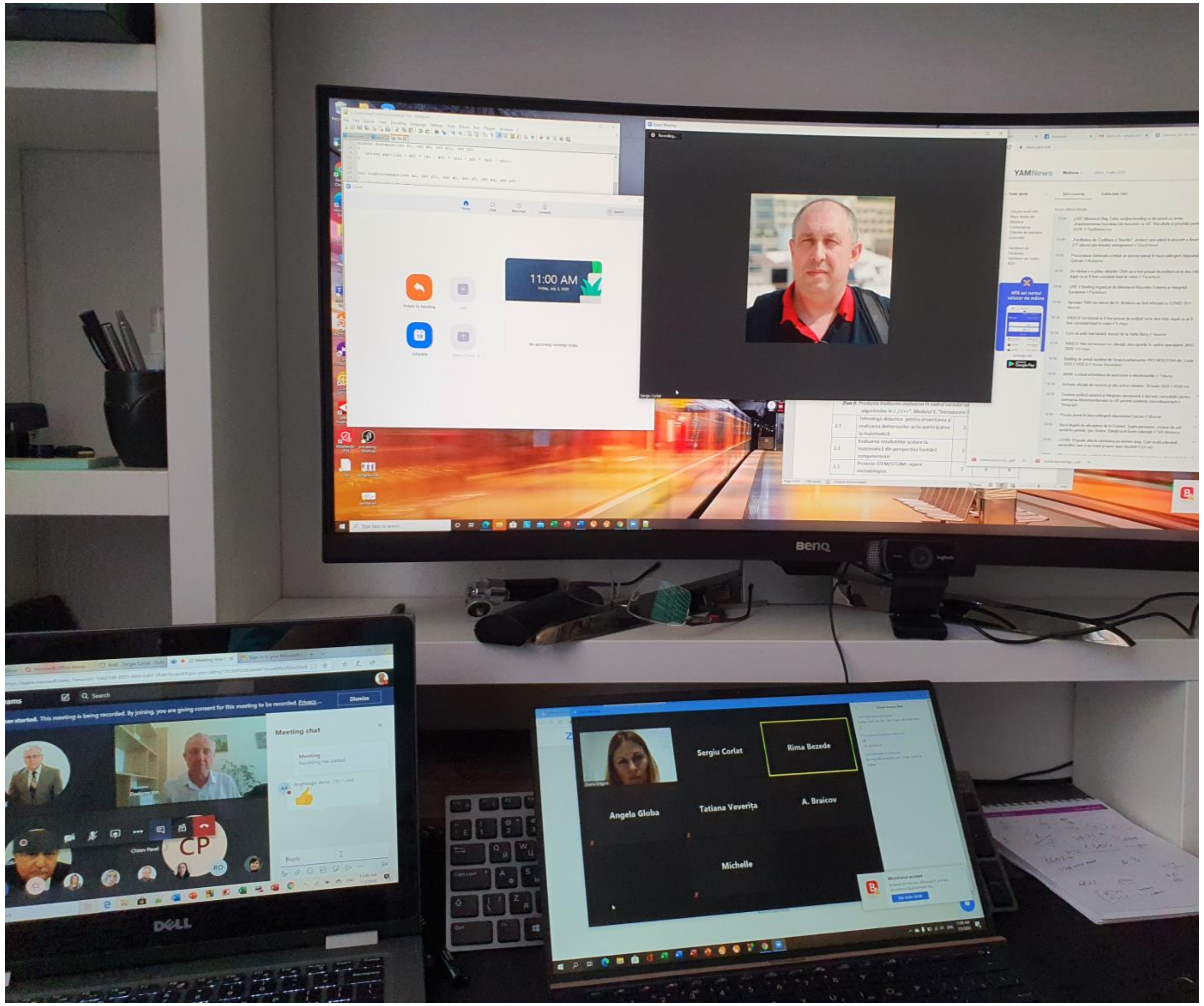
No HDD

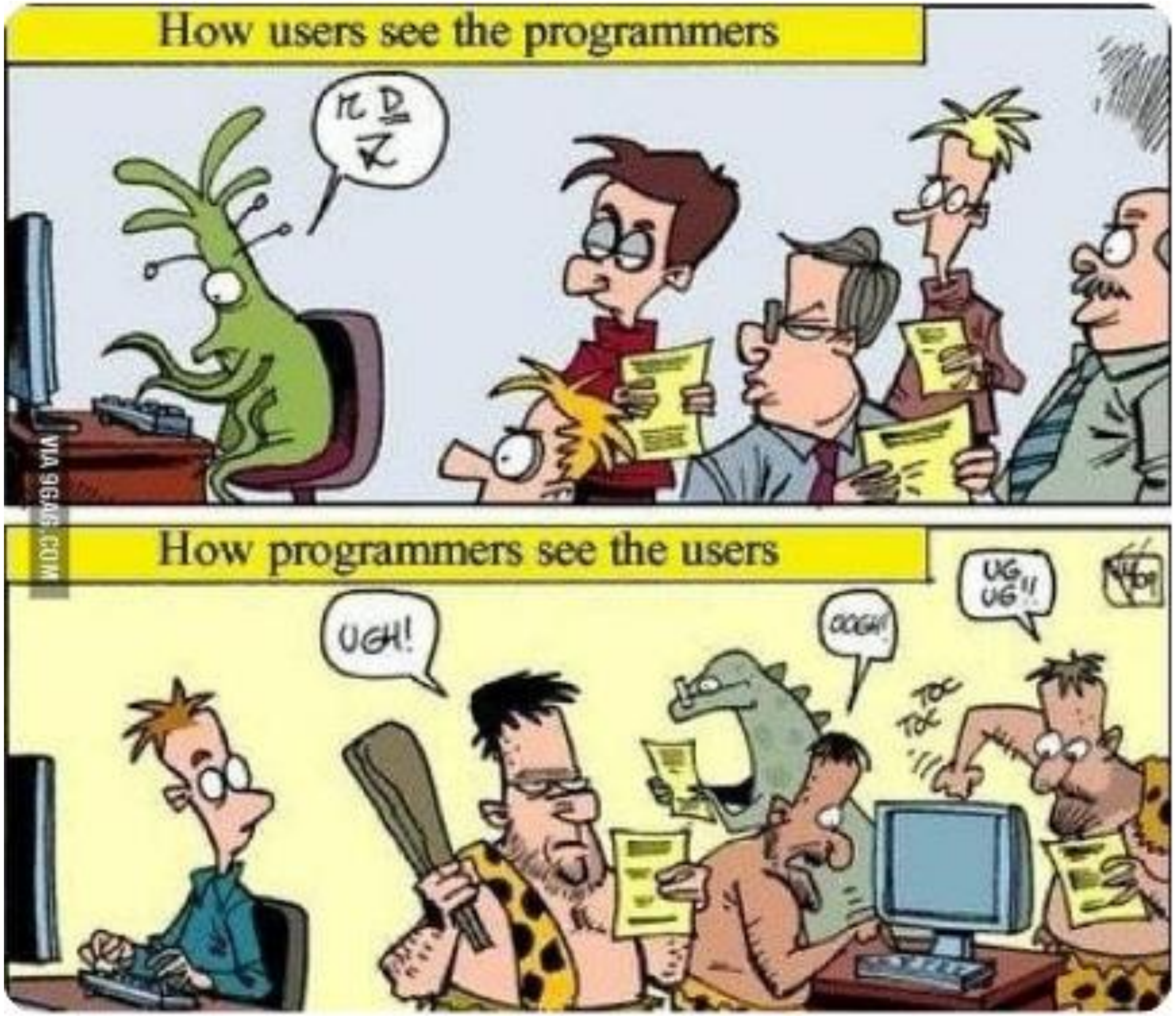
DOS operating system

C compiler (and IDE)

EGA Graphics (480 x 320)

These days working place of an ICT professional




How users see the programmers
How programmers see the users
UGH!
OOGH!
UG UG!!
TOC TOC

That’s all, folks!