Introduction to Computer Science

Lecture 1

Nazareno Aguirre

(Based on material by Dr. Guillaume Hoffmann)

Welcome!

- This course's official name:

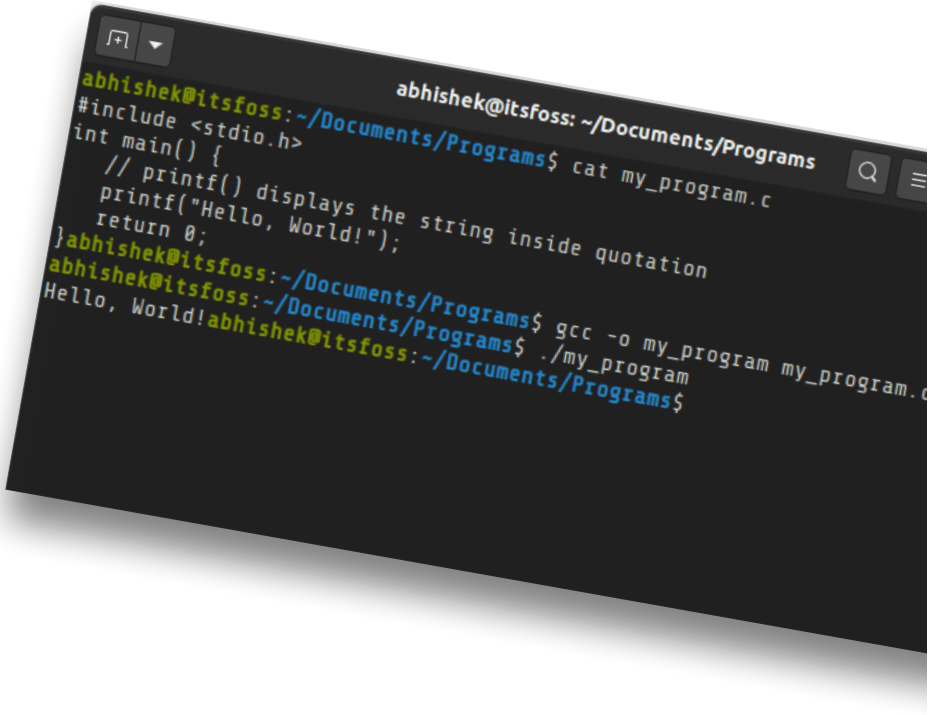
  "Introduction to Computer Science"

- Alternative name:

  *"Learn to **use a computer** to **solve problems through programming (in C)!**"*

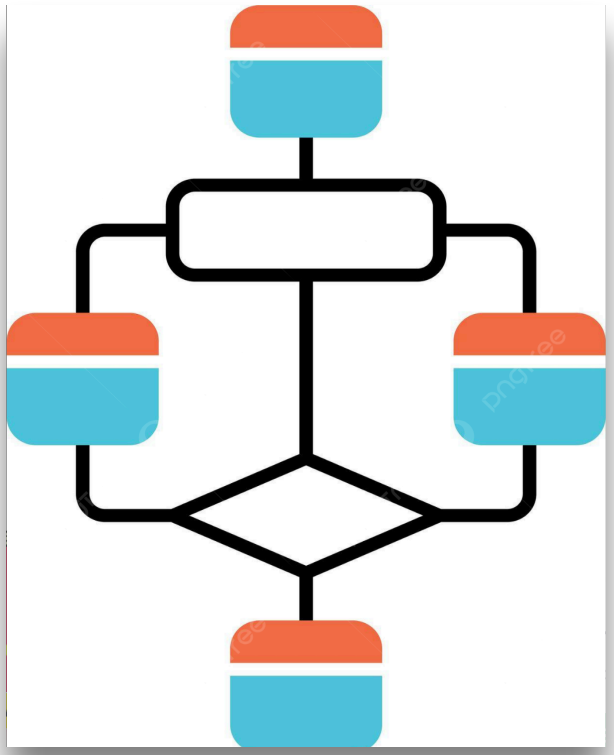
“Use a computer…”

- Learn how to use a computer with a **Linux Operating System**
- Learn how to use a **command-line interface** (text-based, not icon-based)
  - *Why?* because it’s **efficient**
- Learn how to use a basic text editor to write programs
  - We can later on transition to Integrated Development Environments (IDEs)



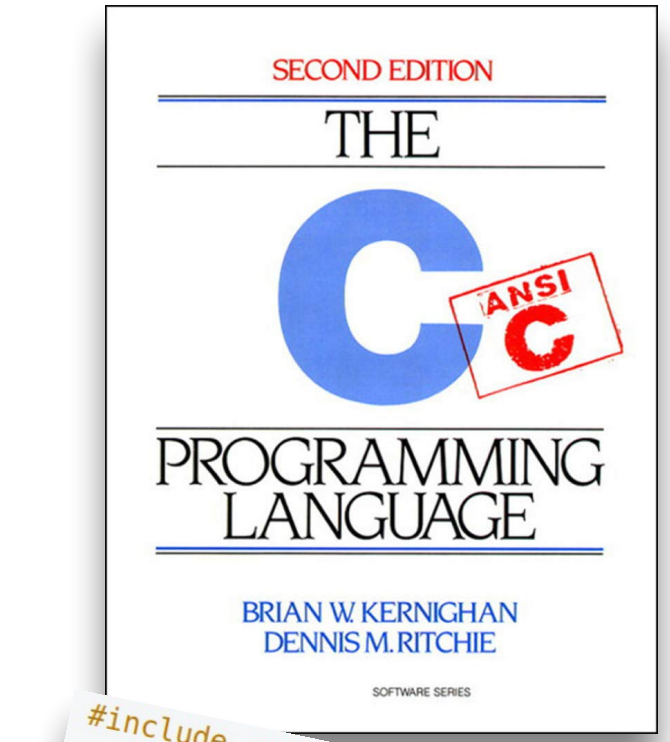
“… to solve problems…”

- This course is not just about technical proficiency
- **The main goal of this course is to learn the basics of algorithmic problem solving**
    - i.e., to automate problem solutions in computer programs
- We will also practice **reading comprehension in English**, to understand problem descriptions and basic programming requirements (exercises)
    - Sometimes exercises will guide you precisely into writing programs. Sometimes you will have to design non-trivial programmatic solutions to solve problems
- Your solution’s worth depends on several criteria but an essential one is: ***does it solve the problem described in the exercise?***



“… through programming (in C)”

- C is the programming language we’ll use in this course, to write programs
- It’s one of the most widely used programming languages
- **Learning programming is a long journey**, but we will progress step by step
- Mastering the whole of C demands a significant effort. We will not learn the whole C language in this course
- C can be complicated but we will strive to **write programs that are simple and clear**



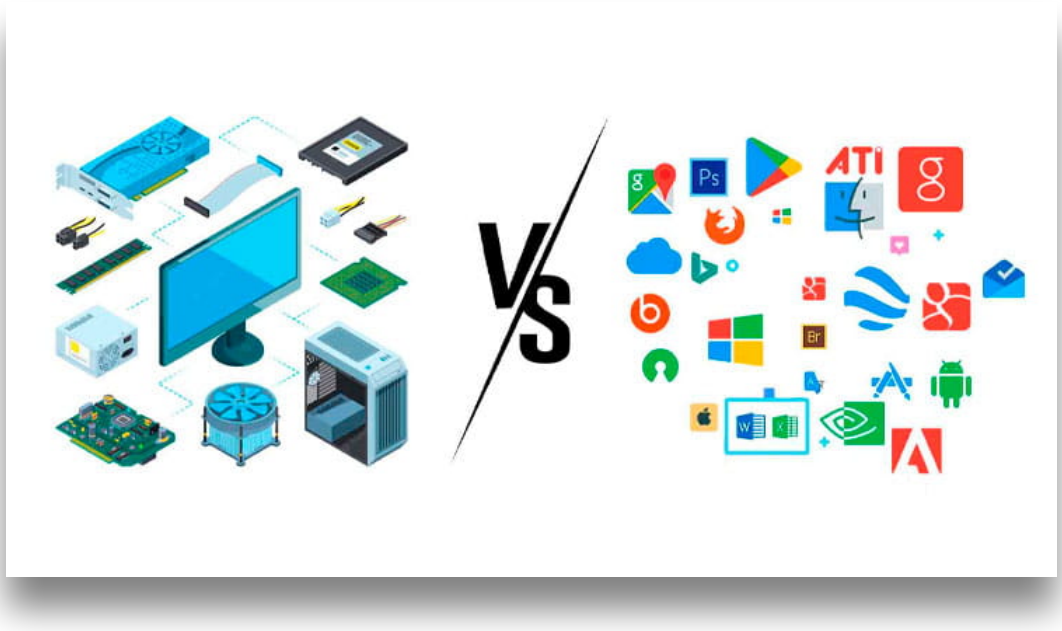

```c
#include <stdio.h>

int main()
{
  printf("Hello, World!\n");
  return 0;
}
```

Computer Systems: Hardware and Software

Hardware and software are the two essential parts of a computer system.

- **Hardware** – all of the physical components of the computer.
- **Software** – stored information (data and instructions) that enables one to operate the hardware.



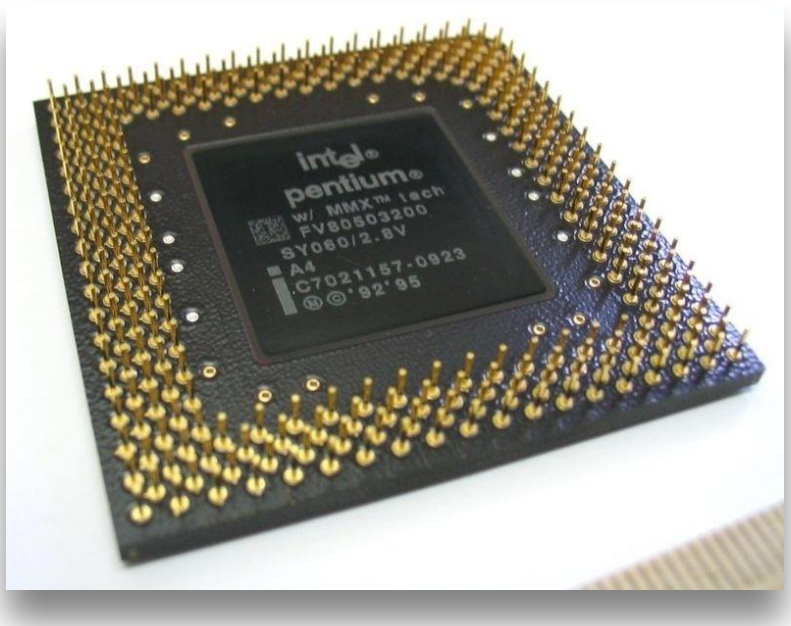
Hardware

All **physical components** of the computer:

- Central Processing Unit (CPU)
- Input/Output components for communicating with the user or other computers.
- Memory for saving data: primary (main) as well as secondary (disks)
- Elements to perform inner communication between all the above.

Central Processing Unit (CPU) or Processor

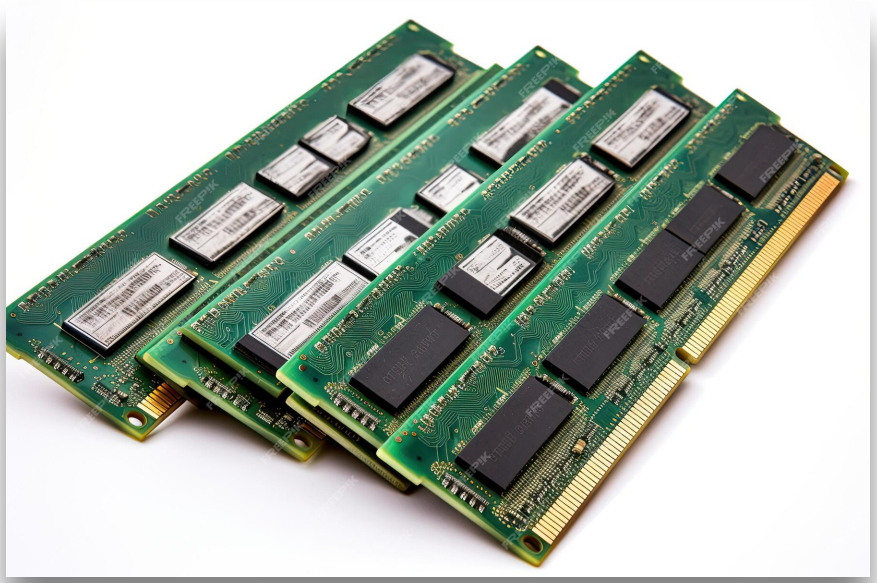
- Main hardware component responsible of program execution
  - It executes very elementary instructions
    - arithmetic calculations: add, subtract, multiply, etc.
    - logical calculations
    - basic decisions upon the data
- It accesses information outside the CPU through data reading and writing



Means of Input and Output

Primary (Main) memory

- Typically called **RAM (Random Access Memory)**
- Fixed into the computer
- Very fast
- Relatively expensive (Most computers have relatively “little” RAM)
- **Volatile** (erased each time the computer is shut down)



Secondary Memory

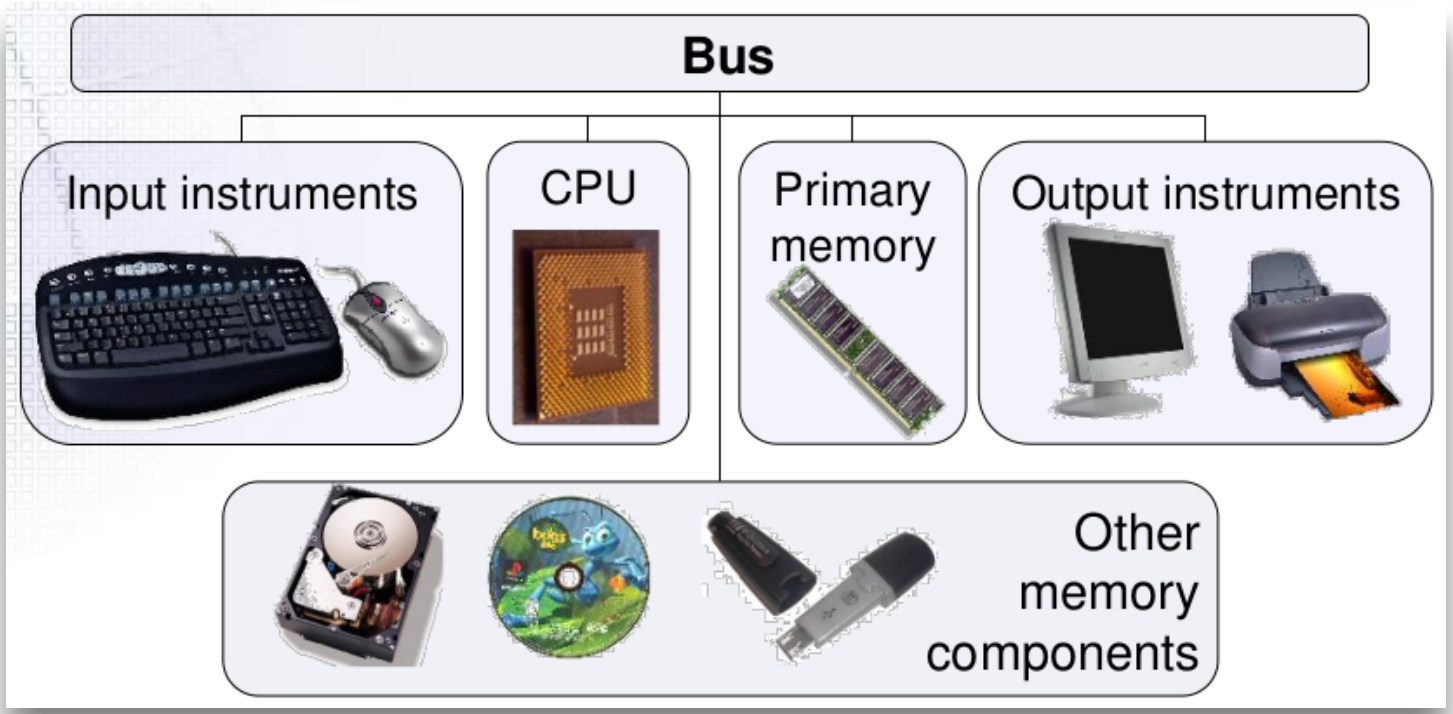
- Most computers also have a **secondary memory**.
- **It is permanent (non-volatile)**: it does not need electric current to store data.
- In “classic” Hard Disks (HD), data is saved by magnetic means
- In Solid State Drives (SSD), data is saved into electronic circuits.
- BIG Volumes: They can easily be 500X bigger than primary memory.
- Relatively slower than primary memory (but SSDs are much faster than HDs).

Removable Memory Media

- Generally, relatively slow
- Volume can change between different memories
- Non-volatile memory
- Main advantage: removable

Communication (bus)

Data is transferred between all the components through a bus



The data in the memory

All memory tools (instruments) save data by different means:

- Electronically (primary memory, USB drives, memory cards)
- Optical (CD, DVD)
- Magnetic (HD, Diskette)

In each case, some physical measure can be in two states:

- High voltage/low voltage; magnetic orientation, etc.
- We interpret these two states states as 0 or 1
- The memory is, in fact, a large series of **binary digits (bits)**, each one having value 0 or 1

Bits and Bytes

Memory is organized in basic groups of **8 bits**, called a **byte**.

How many different states can a byte be in?

1 bit can be in one of 2 states

2 bits can be in one of 2*2 = 4 states

3 bits can be in one of 2*2*2 = 8 states

4 bits can be in one of 2*2*2*2 = 16 states….

… one byte (8 bits) can be in one of 2^8 == **256** different states.
(00000000 to 11111111)

What can be done with bits?

- **The data in a computer memory is stored binary values**
  - Sequences of bits
- Binary values can be straightforwardly mapped back and forth to numeric (decimal) values
- What can be done solely with bits, or equivalently, numbers?
  - A lot!

179	145	159	248	215	237	62	244	63	6	63	175	250	161	182	128
227	252	137	23	189	97	215	203	154	95	46	30	201	102	198	229
229	96	250	216	34	181	163	11	225	53	240	117	124	114	250	255
240	132	47	37	152	141	174	69	53	81	134	36	82	153	100	176
96	249	181	215	40	7	28	226	87	69	61	222	69	5	175	201
49	16	242	28	224	58	155	190	201	140	11	255	74	34	192	82
149	156	28	43	145	90	90	40	40	24	205	1	25	186	159	35
250	207	208	141	137	219	227	130	12	116	148	13	65	140	244	13
91	253	140	142	27	133	81	28	115	239	89	239	47	207	10	157
236	184	77	157	238	242	42	167	179	196	214	170	101	125	236	46
254	58	115	48	151	250	205	92	136	34	5	94	32	236	11	209
110	102	22	36	72	150	92	10	23	204	72	35	254	245	5	105
246	135	181	128	71	208	220	112	19	233	122	236	62	56	164	81

Example

We can encode letters from an alphabet, to encode messages in a language as sequences of numbers:

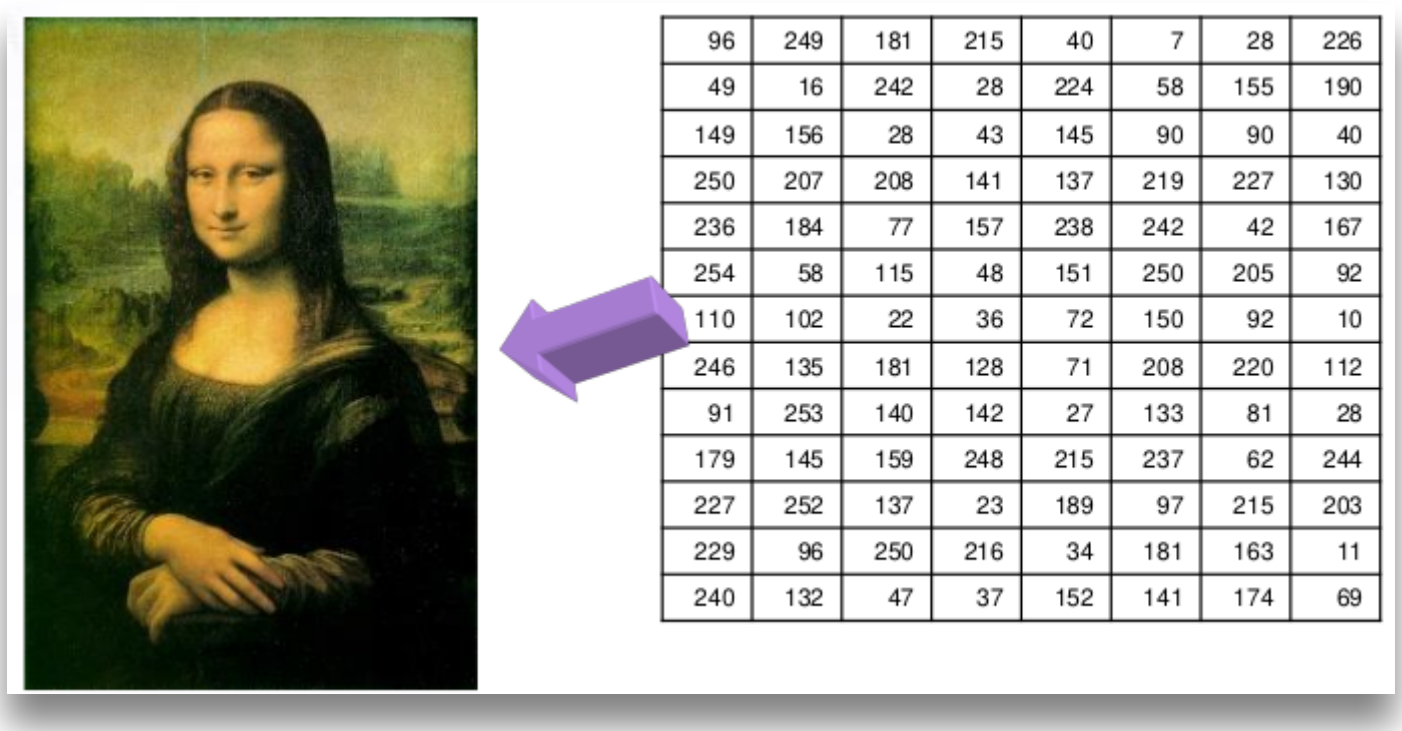
שָׁלוֹם רָב שׁוּבֵךְ, צִפֹּרָה נֶחְמֶדֶת,
מֵאַרְצוֹת הַחֹם אֶל-חַלּוֹנִי-
אֶל קוֹלֵךְ כִּי עָרֵב מַה-נַּפְשִׁי כָּלָתָה
בַּחֹרֶף בְּעָזְבֵךְ מְעוֹנִי.

זַמְּרִי, סַפֵּרִי, צִפּוֹרִי הַיְקָרָה,
מֵאֶרֶץ מֶרְחַקִּים נִפְלָאוֹת,
הֲגַם שָׁם בָּאָרֶץ הַחַמָּה, הַיָּפָה,
תִּרְבֶּינָה הָרָעוֹת, הַתְּלָאוֹת?

179	145	159	248	215	237	62	244
227	252	137	23	189	97	215	203
229	96	250	216	34	181	163	11
240	132	47	37	152	141	174	69
96	249	181	215	40	7	28	226
49	16	242	28	224	58	155	190
149	156	28	43	145	90	90	40
250	207	208	141	137	219	227	130
91	253	140	142	27	133	81	28
236	184	77	157	238	242	42	167
254	58	115	48	151	250	205	92
110	102	22	36	72	150	92	10
246	135	181	128	71	208	220	112

Example

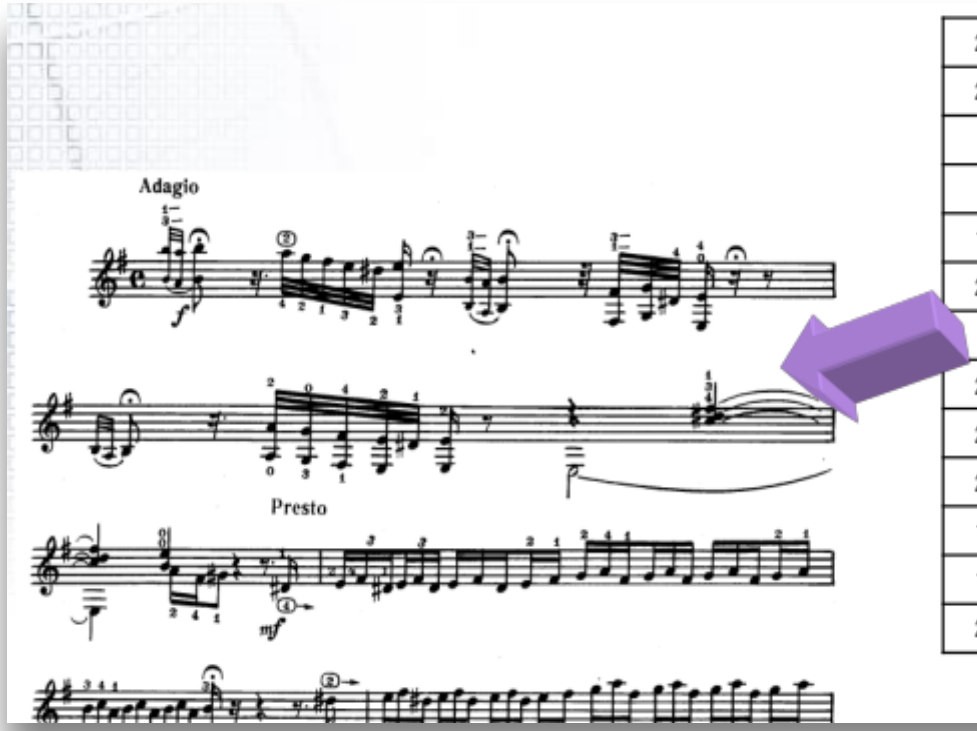
We may also use numbers as representations of colors. Then, a picture or photo may be thought of as a sequence of numbers, the number encoding the color of each single pixel in the picture



96	249	181	215	40	7	28	226
49	16	242	28	224	58	155	190
149	156	28	43	145	90	90	40
250	207	208	141	137	219	227	130
236	184	77	157	238	242	42	167
254	58	115	48	151	250	205	92
110	102	22	36	72	150	92	10
246	135	181	128	71	208	220	112
91	253	140	142	27	133	81	28
179	145	159	248	215	237	62	244
227	252	137	23	189	97	215	203
229	96	250	216	34	181	163	11
240	132	47	37	152	141	174	69

Example

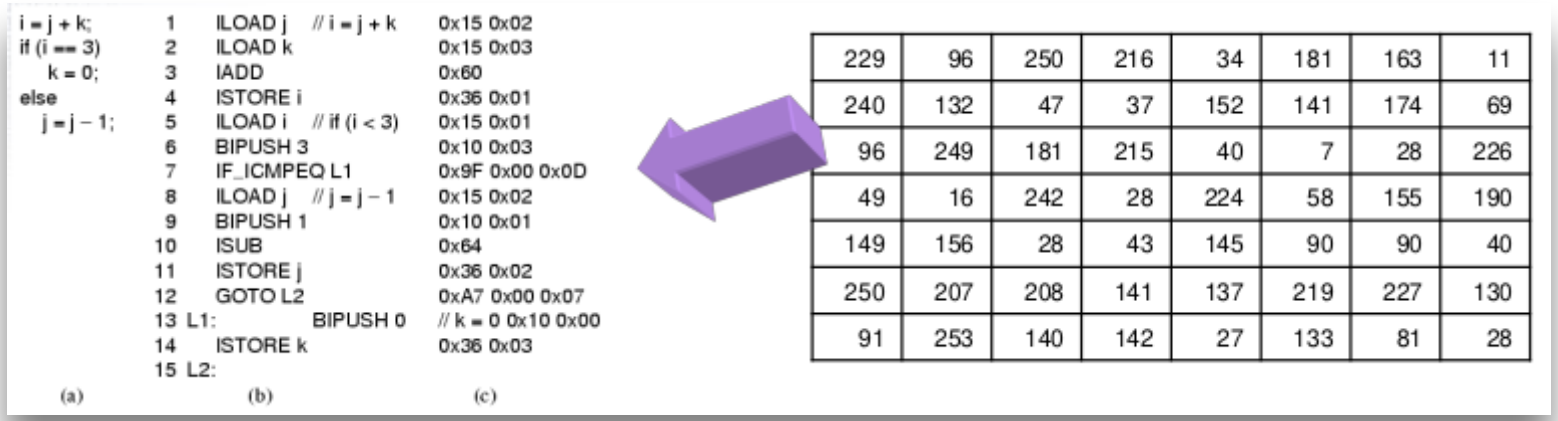
We can also use numbers, e.g., to represent musical notes:



229	96	250	216	34	181	163	11
240	132	47	37	152	141	174	69
96	249	181	215	40	7	28	226
49	16	242	28	224	58	155	190
149	156	28	43	145	90	90	40
250	207	208	141	137	219	227	130
91	253	140	142	27	133	81	28
236	184	77	157	238	242	42	167
246	135	181	128	71	208	220	112
254	58	115	48	151	250	205	92
110	102	22	36	72	150	92	10
179	145	159	248	215	237	62	244
227	252	137	23	189	97	215	203

Code as numbers

- Numbers also can be interpreted as commands to be executed (run) by the computer.
- For example :
  - 0 – read a number from memory.
  - 1 – add 2 numbers.
  - 2 – subtract between 2 numbers
  - …
- Each CPU has its own particular way of understanding commands. This is the so called **machine language**.



```
i = j + k;                1   ILOAD j    // i = j + k     0x15 0x02
if (i == 3)               2   ILOAD k                    0x15 0x03
    k = 0;                3   IADD                       0x60
else                      4   ISTORE i                   0x36 0x01
    j = j - 1;            5   ILOAD i    // if (i < 3)    0x15 0x01
                          6   BIPUSH 3                    0x10 0x03
                          7   IF_ICMPEQ L1                0x9F 0x00 0x0D
                          8   ILOAD j    // j = j - 1     0x15 0x02
                          9   BIPUSH 1                    0x10 0x01
                         10   ISUB                        0x64
                         11   ISTORE j                    0x36 0x02
                         12   GOTO L2                     0xA7 0x00 0x07
                         13 L1:          BIPUSH 0        // k = 0 0x10 0x00
                         14   ISTORE k                    0x36 0x03
                         15 L2:
          (a)                    (b)                          (c)
```

229	96	250	216	34	181	163	11
240	132	47	37	152	141	174	69
96	249	181	215	40	7	28	226
49	16	242	28	224	58	155	190
149	156	28	43	145	90	90	40
250	207	208	141	137	219	227	130
91	253	140	142	27	133	81	28

Programming in machine language is difficult

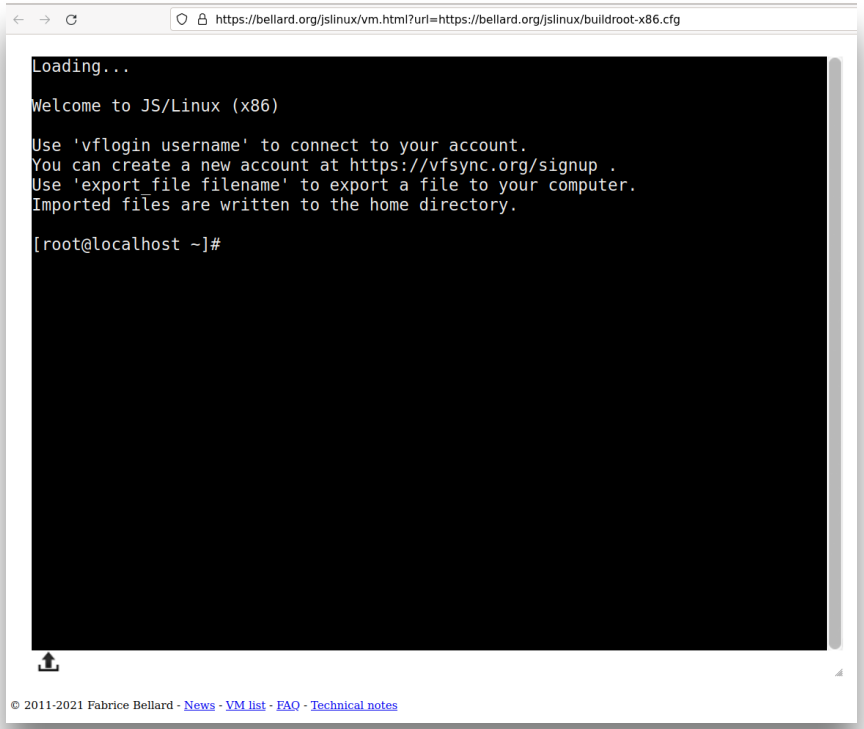
- Every computer architecture has its own machine language. The definition of this language cannot be changed.
- Machine language is designed for the computer to execute it
  - Writing relatively simple programs in machine language can be highly cumbersome
    - Also reading and understanding such operations is difficult and costly
- Thus, programmers usually use **high level languages**.
    - C, Python, Java, …
- A special tool will **translate our high level programs into a machine language**, so that the machine can understand it and run it: **the compiler**.

Translating from high level languages to machine code

- A **compiler** is a software tool that **automates the translation from a high level programming language to machine code.**
- The C programming language, that we will use in this course, is an example of high level programming language
- We will need a C compiler to translate our C programs to machine language, and execute the resulting code.
- Programs in high level programming languages, like C, are **portable**
    - They can be run on different platforms/architectures, as long as we have compilers for such platforms
- Compilers also typically improve code efficiency in the translation, to profit from characteristics of the target computer architecture
- Compilers are also able to **detect syntactic errors** in our programs, allowing us to identify programming issues
- There exist various alternative C compilers
    - We will use one of the most notable: GCC

JSLinux: a computer in your web browser!

- Open this link from your laptop/ desktop computer (no phone/tablet):
- https://bellard.org/jslinux/vm.html?url=https://bellard.org/jslinux/buildroot-x86.cfg
- Or from https://bellard.org/jslinux/ select the first option (Alpine Linux 3.12.0, Console, click on startup link)
- What do you see?



Some Sample Commands

- **uptime** : tells how much time passed since the system started
- **uname** : tells the name of the operating system
- **uname -a** : same, but gives more details
- **pwd** : tells the path of the current directory
- **ls** : shows the list of files in the current directory
- **ls -l** : same, but with more details
- **cat hello.c** : shows the contents of the file **hello.c** of the current directory

How to execute a command

- Type the command exactly as it is spelled
- If the command comes as a list of words, put at least one space between each word
- Type the Enter key

Examples of command structure

Single command:

uptime

Command with parameter (tells **to which** object apply the command):

cat hello.c

Command with flag (tells **how** the command should be executed):

ls -l

Command with flag and parameter:

ls -l directory

Navigate in the file system

- Files are organized as a hierarchy or directories and files.
- Directories behave like sets in mathematics. Directories and files can be elements.
- A file or directory is referred by its **path**.
- In Linux the root directory is **/**
- Example of path of a file: **/home/aguirre/to-do.txt**
- A path that starts from the root directory is called **absolute path.**
- If it does not, it is called a **relative path** (to the current directory)**.**

The current directory

- Command **pwd** shows you the path of the current directory.
- **cd** is a command that makes you change the current directory (that is, you “travel” to another directory):
- **cd** without parameter makes you go to your home directory
- **cd ..** makes you go to the parent directory
- **cd dir111** makes you go to the child directory **dir111** (if it exists)

More commands related to files

- **cp** : copy a file
- **cp -r** : copy a directory
- **mv** : rename or move a file or a directory
- **rm** : delete a file
- **rm -r** : (recursively) delete a directory
- **mkdir** : create a directory
- Tip: in the command line interface, you can type the beginning of some file name and **press TAB**, if there is no ambiguity the name will be autocompleted.

Compile and execute our first C program in JSLinux

- JSLinux comes with a few files and directory (when you reset the system they reappear)
- **hello.c** is a file that contains the source code of a simple “Hello world” C program
- See its content by executing **cat hello.c**
- To compile and execute, you may use gcc:
- **gcc hello.c -o hello** : program in file hello.c is compiled using **gcc**, into a file named **hello**
- **./hello** : you execute the compiled program (it is necessary to type **./** at the beginning)