COMPUTER BASICS:

CPU = Central Processing Unit
system memory =
***RAM* =** random access memory
Hard drive
I/O = input/output aka screen, mouse, keyboard

everything is hooked together via a main board or motherboard composed of the different element and their pathway

classes of system:
Client/workstation = PC = single user device
Server = respond to request from network = no direct human interface, use of server a network to communicate with it
Hand-held = phone, audio player
embedded system = toaster/car

RAM:
it stores bits, organized by bytes (1 byte = 8bits) address starts from 0, the CPU uses the address to read the data stored at a given place
> **addressable by the CPU**
> **volatile :**
when the power goes off the content of the ram gets erased
> **faster than storage to read and right**
> **stores code and data of running programs**

CPU/ programming model: executes binary instructions, the cpu reads one instrction after the other and performs one of the following tasks:
> **copy bytes**
> **aritmetic**
at least addition and negation
> **bit logic**
not / and/or / exclusive or
(those single instructions help manipulating single bits amongst bytes)
> **jumps**
navigating memory
the CPU is composed of registers, two kinds:
> **status:** stores data affecting the operation of the CPU

> **general purpose:** stores any data because the CPU computes inside itself, needed data to perfom operation is briefly stored there

ISA = Instruction Set Architecture: **x86** (pcs Intel and AMD produced) / **ARM** for phones

CPU --read/wrie→((registers)device)
CPU ->(USB controller)-------(usb keyboard/usb drive/usb mouse)

OPERATING SYSTEM BASICS:

> load and manage processes
When the computer starts it runs an os, its a program to manage the hardware and the running programs. In os terminology a program is called a process. The os keeps the processes from interfering with one another even if they run at the same time.

> provide interfaces to hardware via system calls
It provides an interface to the hardware for processes the program communicate with the hardware via system calls provided by the OS.
> provide filesystem
Abstracts over the storage devices so that the processes can read and write files
> provide a basic user interface
Graphic interface for user to manage programs and files
>Program memory
The CPU can only execute one program at a time, a portion of the os called program scheduler tells who gets to run at a given time
>Device drivers
Plug in module that handles the management of a given i/o.

1 byte = 8 bits of data

Boolean = true or false = 1 or 0

PROGRAM

A program is a sequence of instructions that tell a computer how to do a task. When a computer follows the instructions in a program, we say it *executes* the program. You can think of it like a recipe.

IDE

integrated developpement environement : set up tool that helps write and debug

C

created around 1972 by Kem Thompson and Dennis Ritchi, developped concurrently with Unix
The US wrote a serie of convention C89, C90, C99, C11
Compilers:
GCC (GNU compiler collection)

Clang / LLVM (low level virtual machine)
MSVC (microsoft visual C++)

Imperative and procedural
Statically typed (everything as to be stated, types, returns etc)
Weakly typed (not memory safe)
primary goal is high performance

- C programming language can be used to design the system software like operating system and Compiler.
- To develop application software like database and spread sheets.
- For Develop Graphical related application like computer and mobile games.
- To evaluate any kind of mathematical equation use c language.
- C programming language can be used to design the compilers.
- UNIX Kernal is completely developed in C Language.
- For Creating Compilers of different Languages which can take input from other language and convert it into lower level machine dependent language.
- C programming language can be used to design Operating System.
- C programming language can be used to design Network Devices.
- To design GUI Applications. Adobe Photoshop, one of the most popularly used photo editors since olden times, was created with the help of C.

Operating Systems, Embedded Systems, Real-time

Key Features of Procedural Programming

The key features of procedural programming are given below:

- **Predefined functions:** A predefined function is typically an instruction identified by a name. Usually, the predefined functions are built into higher-level programming, but they are derived from the library or the registry, rather than the program. One example of a pre-defined function is ‘charAt()’, which searches for a character position in a string.

- **Local Variable**: A local variable is a variable that is declared in the main structure of a method and is limited to the local scope it is given. The local variable can only be used in the method it is defined in, and if it were to be used outside the defined method, the code will cease to work.
- **Global Variable**: A global variable is a variable which is declared outside every other function defined in the code. Due to this, global variables can be used in all functions, unlike a local variable.
- **Modularity**: Modularity is when two dissimilar systems have two different tasks at hand but are grouped together to conclude a larger task first. Every group of systems then would have its own tasks finished one after the other until all tasks are complete.
- **Parameter Passing:** Parameter Passing is a mechanism used to pass parameters to functions, subroutines or procedures. Parameter Passing can be done through 'pass by value', 'pass by reference', 'pass by result', 'pass by value-result' and 'pass by the name'.

Concepts:

Scope: a variable is only usable within the curly braces it's been declared into

Procedural:
This paradigm uses a linear top-down approach and treats data and procedures as two different entities.

Actions:

Control of memory: >explicitly manipulate individual bytes = more efficient, urge to clean created elements after we're done with them
>manual allocation

declaring / a declaration : a statement to signify the existence of an entity
definition: statement defining what the entities holds

```
11
10 int»»···function_name(int paramter0, char parameter1); //declaration
9
8
7 int»»···function_name(int paramter0, char parameter1)
6 {
5 »···//the brackets hold the definition of the function
4 »···int»»···var; //variable declaration
3
2 »···var = 15; //definition - initialiazing variable0
1 »···return (variable);
12 }
```

conditionning / conditions: evaluates a situation to execute a given part of the code according to the answer

if true: we will execute the branch that depends on that evaluation
if false: we will jump to the res of the code

the branch starts after the opening curly brace and ends at the closing one

```
9 int»»···f_add(int a, int b)
8 {
7 »···if (a == 0)
6 »···»···return (b);
5 »···if (!b)
4 »···»···return (a);
3 »···return (a + b);
2 }
1
```

loops: condition that will help perform a given action multiple times

exercise: write “Hello Wolrd!” 5 times // factorial

control flow statement:

> continue: jumps to the next conditioninside the loop
> break: exits the loop
> return: exits the function (+ value)

Entities:

Pre processor statement:

A preprocessor statement i s a the top of every C/CPP file of a project. It provides the needed function prototype along with the #includes and the macros and structures etc. What the compiler does is that it copies the content of the header at the top of every file before turning everything into a binary .exe .

> finds a named file and pastes it in current file
> gets evaluated before processing

```
2 #include <native_header.h>
1 #include "custom_header.h"
```

header files:
traditionally used to declare functions that are located in other files for them to be found when invoked from external files

only stores declarations

ex: add function

variable: naming a piece of data to store it in memory
(> store player position on map, health point etc)

primitive data type: each has a specific purpose

the real difference between variables is the size it holds in the memory (char = 1oct = 1byte, int = 4bytes = 32bit…)

how to use a variable:

declare it type + name + assigning value (initializing)

sizeof(variable) will give you the precise size of the given type

```
11
10 int»»···function_name(int paramter0, char parameter1); //declaration
 9
 8
 7 int»»···function_name(int paramter0, char parameter1)
 6 {
 5 »···//the brackets hold the definition of the function
 4 »···int»»···var; //variable declaration
 3
 2 »···var = 15; //definition - initialiazing variable0
 1 »···return (variable);
12 }
```

Function: Block of code designed to perfom a given task > methods

most of the time it has an input and an outpout but it can run without them

```
int»»···f_add(int a, int b)
{
»···int»»···res; //declaring result

»···res = a + b;
»···return (res);
}
```

Funtcion prototype:

A prototype is composed of, from left to right, the return value, the name of the function, the input parameters.

```
int»»···function_name(int paramter0, char parameter1);
return_type»»···fuction_name(parameter_type variable_name)
```

pointers: integer holding an address

exercise: swap

Structures: class without visibility and methods