Computers
Data Representation

Grau en Ciència i Enginyeria de Dades

Xavier Martorell

Xavi Verdú

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC)

2019-2020 Q2

Creative Commons License

Table of Contents

Basic Concepts

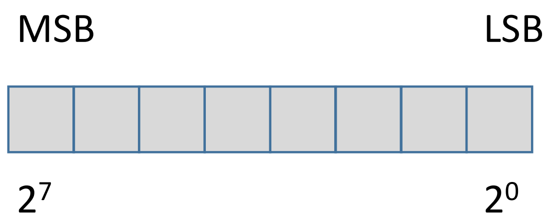
- Binary Digit: it is a number that can adopt two values: 0 or 1
  - Many interpretations: true/false, positive/negative, on/off, etc

- Bit: contraction from “Binary Information Digit” *(John W. Tukey, 1947)*
  - It is the maximum amount of information that can be conveyed by a binary digit
    - A binary digit can convey between 0 and one bit of information

- Byte: it is a unit of digital information, usually comprising 8 bits
  - The smallest addressable unit of memory in many, but not all, computer architectures
    - Introduced in the ‘60s, and popularized by Intel (8008) in the ‘70s
    - It was used to encode a text character

Binary number using 8-bit Byte

- MSB: Most Significant Bit
- LSB: Least Significant Bit



- Example:
  - Decimal number: 23
  - Binary representation: ???

Signed Number Representations

- A binary number can represent negative values using different methods
  - MSB = 1 represents negative numbers
  - Optimize substractions
    - E.g. add negative numbers

- The most known and used methods are:
  - Signed and magnitude
    - Commonly used to represent the mantissa of floating points (see later slides)
  - One’s Complement
    - Early computers and other current particular usages
  - Two’s Complement
    - Most of current computers
  - Biased representation
    - Exponent of floating points (see later slides)

Two’s Complement

- Range:
  - From $-2^{(N-1)}$ to $+(2^{(N-1)}-1)$

- Advantage compared to one’s complement
  - Disregard overflow bit in operations (e.g. substraction)

- Negative number:
  - Decimal number -> invert bits -> add +1

- Example:
  - Decimal number: -23
  - Two’s Complement representation: ???

Basic Concepts

- Data type is a classification that let's the interpreter/compiler knows how the data is going to be used

- Direct impact on size and format

- Software and hardware support
  - Compiler/Interpreter and architecture
    - 32-bit vs 64-bit

Basic Classification

- **Scalar data types**: a single value
  - Arithmetic (numbers), symbols and characters, boolean, enumeration, pointers
- **Special data type**: Void
- **Compound/Aggregate data types**: built combining one or more scalar types
  - Arrays, structures, unions

Basic Background

- Endianness
  - The order of bytewise values in memory

E.g. $(1234)_{10} = (04\ D2)_H$

- Big-Endian
  - Byte with **most** significant value: stored first (lowest memory address)
  - Data networking and mainframes

@	Value
0x0000	**04**
0x0001	D2

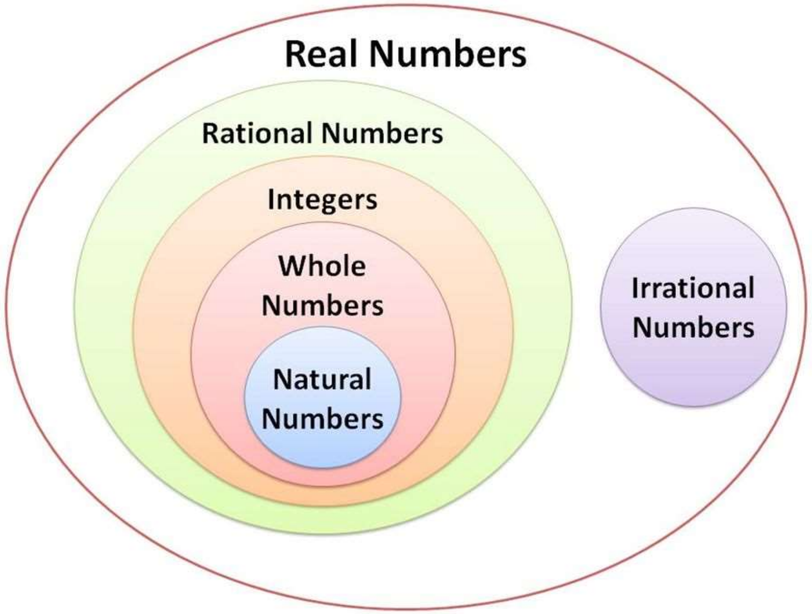
- Little-Endian
  - Byte with **least** significant value: stored first (lowest memory address)
  - x86 Intel processor family and most microprocessors

@	Value
0x0000	**D2**
0x0001	04

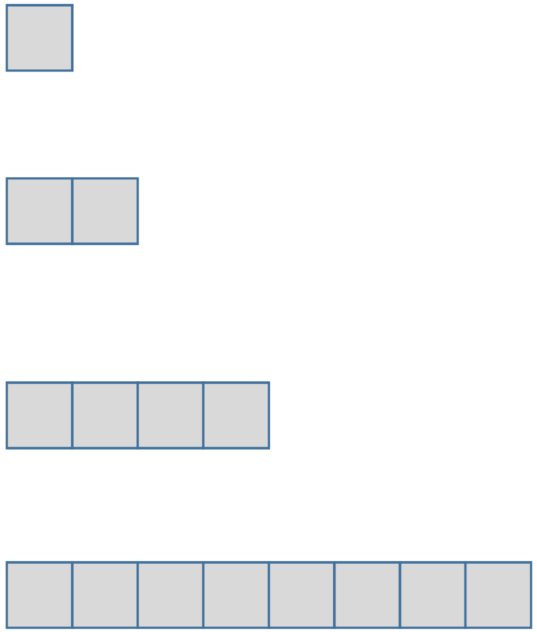
- Some architectures support both
  - E.g. ARM, recent x86 and x86-64

Scalar Data Type: Arithmetic (numbers)

- Integer and real numbers
- Signed and unsigned data types
- Numeral Systems
    - Binary, Octal, Decimal, Hexadecimal



Integers



- Char: 1 Byte
  - Signed: from -128 to 127
  - Unsigned: from 0 to 255
- Short: 2 Bytes
  - Signed: from −32,768 to 32,767
  - Unsigned: from 0 to 65,535
- Integer: 4 Bytes
  - Signed: from -2,147,483,648 to 2,147,483,647
  - Unsigned: from 0 to 4,294,967,295
- Long Long: 8 Bytes (if possible)
  - Signed: from –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  - Unsigned: from 0 to 18,446,744,073,709,551,615

- Different base notation
  - Decimal, octal, hexadecimal

Integer versus Long Integer

- Long integer depends on both architecture Operating System

OS	Arch	Size (Bytes)
Windows	IA-32	4
Windows	x86-64	**4**
Linux	IA-32	4
Linux	x86-64	**8**
Mac OS X	IA-32	4
Mac OS X	x86-64	**8**

- Warning when porting code
  - **The behavior may change**
  - E.g.: IA-32 vs x86-64 Linux, x86-64 Win vs x86-64 Linux

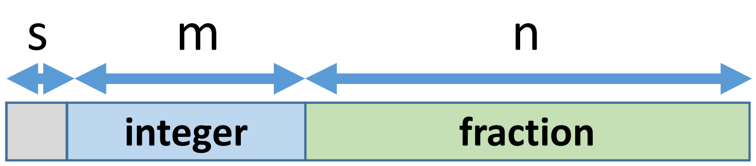
https://software.intel.com/en-us/articles/size-of-long-integer-type-on-different-architecture-and-os

Real Numbers

- Numbers with a fractional component
- Two main representations
  - Fixed-point versus Floating-point
    - The radix point is fixed or can float anywhere
      - The symbol to separate integer and fractional parts of a real number
    - The lesson “computer and its elements” will introduce the cost difference among Hw supports
- Implementation: tradeoff between cost and precision
  - Lack of hardware resources
    - E.g.: Multimedia decoders
  - Boost performance although degraded precision
    - E.g. Playstations, Doom

Fixed-point numbers

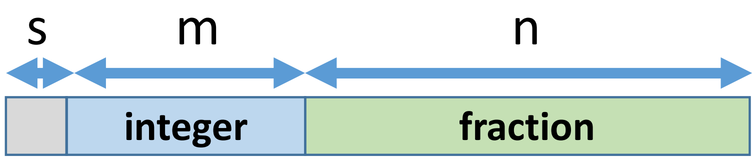
- Bits = 1 + m + n
  - 1 bit for sign (if signed)
  - m bits for integer component
  - n bits for fraction component



- Notation: $Q_{m.n}$
  - Integer number without fraction component ($Q_{m.0}$)
  - Fractional number without integer component (Q_n)

Fixed-point numbers

- **Value = $-2^m b'_s + 2^{m-1} b'_{m-1} + \ldots + 2^1 b'_1 + 2^0 b'_0 + 2^{-1} b_{n-1} + 2^{-2} b_{n-2} + \ldots + 2^{-n} b_0$**
- Programming language support
  - C and C++ have no direct support, but can be implemented
    - Embedded-C supports it (implemented in GCC)
  - Python has direct support through a module
    - I.e. decimal module



Example: 1110

$Q3.0$: $-2^3 + 2^2 + 2^1 = -2$

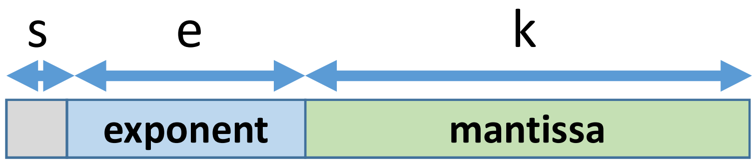
$Q1.2$: $-2^1 + 2^0 + 2^{-1} = -2 + 1 + 0.5 = -0.5$

$Q3$: $-2^0 + 2^{-1} + 2^{-2} = -1 + 0.5 + 0.25 = -0.25$

Fixed-point numbers: accuracy problems

- Precision loss and overflow
- Results can require more bits than operands
  - Round or truncate
  - Specify different size for result
- Boundary numbers to prevent overflow
- Exception: overflow flag
  - If supported by hardware

Floating-point numbers

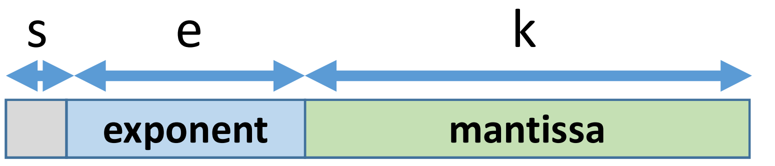


- Bits = 1+ e + k
  - 1 bit for sign (if signed)
  - e bits for exponent {1, …, (2^e-1)-1}
  - k bits for mantissa (fraction)
    - There is an implicit 1-bit (top left) equals to 1, **unless exponent is equal to zero**

- Most processors follow IEEE floating point standard
  - First version on 1985
  - Standardize formats
  - Special Values

Floating-point numbers

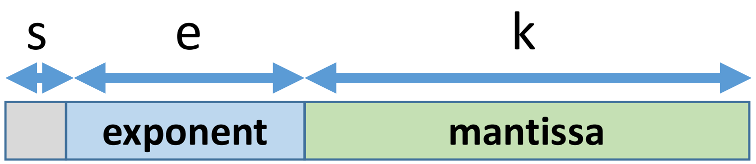
- **Value = $(-1)^{sign} * (1 + \sum b_{(k-i)}\, b^{-i}) * b^{(e-(Emax))}$**



- Float: 4 Bytes
  - Sign bit, 8-bit exponent, 23-bit mantissa
- Double: 8 Bytes
  - Sign bit, 11-bit exponent, 52-bit mantissa
- Some languages support 10 bytes
  - Sign bit, 15-bit exponent, (1+63)-bit mantissa

Floating-point numbers

- **Intel encoding**



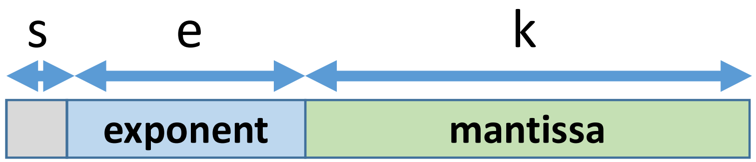
	Sign	Biased exp.	Magnitude	
Single precision		8 bits	23 bits	(1+31)
Double precision		11 bits	52 bits	(1+63)
Double extended precision		15 bits	**1**+63 bits	(**1**+1+78)

https://software.intel.com/en-us/articles/x87-and-sse-floating-point-assists-in-ia-32-flush-to-zero-ftz-and-denormals-are-zero-daz

Class		Sign	Exp.	Mantissa
Positive	+∞	0	11 … 11	1 . 00 … 00
	+Normals	0 … 0	11 … 10 … 00 … 01	1 . 11 … 11 … 1 . 00 … 00
	+Denormals	0 … 0	00 … 00 … 00 … 00	0 . 11 … 11 … 0 . 00 … 01
	+Zero	0	00 … 00	0 . 00 … 00
Negative	-Zero	1	00 … 00	0 . 00 … 00
	-Denormals	1 … 1	00 … 00 … 00 … 00	0 . 00 … 01 … 0 . 11 … 11
	-Normals	1 … 1	00 … 01 … 11 … 10	1 . 00 … 00 … 1 . 11 … 11
	-∞	1	11 … 11	1 . 00 … 00
NaN	SNaN	X	11 … 11	1 . 0X … XX
	QNaN	X	11 … 11	1 . 1X … XX
	QNaN	1	11 … 11	1 . 10 … 00

Floating-point numbers

- **Value = $(-1)^{sign} * (1 + \sum b_{(k-i)}\, b^{-i}) * b^{(e-(Emax))}$**



Example: $(23{,}46875)_{10} = (23)_{10} + (0{,}46875)_{10} =$

$= (10111)_2 + (0.01111)_2 = (10111.01111)_2 =$

$= (1.011101111)_2 * 2^4$

32-bit floating point representation

Sign (1-bit) = 0

Exponent (8-bit) = $4 = (131 - E_{max}) = (131 - 127)$

$= (10000011)_2$

Mantissa (23-bit) = (0111011110…0)

0 10000011 01110111100000000000000

$(41BB\ C000)_H$

Floating-point numbers: support

- Float
  - C and C++: single-precision (32-bit)
  - Python: built-in double-precision (64 bit)

- Most 32-bit architectures comprises 64-bit support in FPU (floating-point unit)
  - IA-32 and x86-64 present 80-bit floating-point type (double-extended precision format)
    - From 1989: x87 FPU (80-bit)

- Quad-precision (128-bit)
  - Software support
  - Few architectures provide hardware support
    - E.g. IBM POWER9 processors (MareNostrum 4)

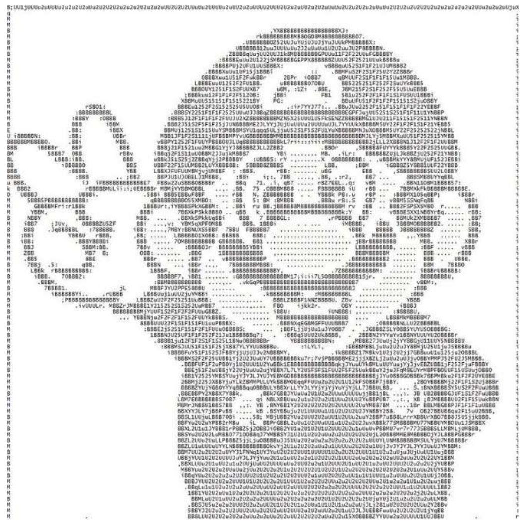
Floating-point numbers: accuracy problems

- Numbers that cannot be exactly represented as binary fractions
  - E.g. 1/10
    - 0.00011001100110011001100110011001100110011001100110011001…
- Conversion to integer loses acuracy due to truncate and roundoff
  - E.g. 56 / 7 = 8;   0,56 / 0,07 = can be 7
    - E.g. explosion of Ariane 5 rocket (1996)
      - http://www-users.math.umn.edu/~arnold/disasters/ariane.html
- Conmutative, but not necessary: associative and distributive
  - “(a + b) + c” could be not equal to “a + (b + c)”
  - “(a + b) * c” could be not equal to “a * c + b * c”

Scalar Data Type: Symbols and Characters

- Char data type
  - Encode alphanumeric data and symbols

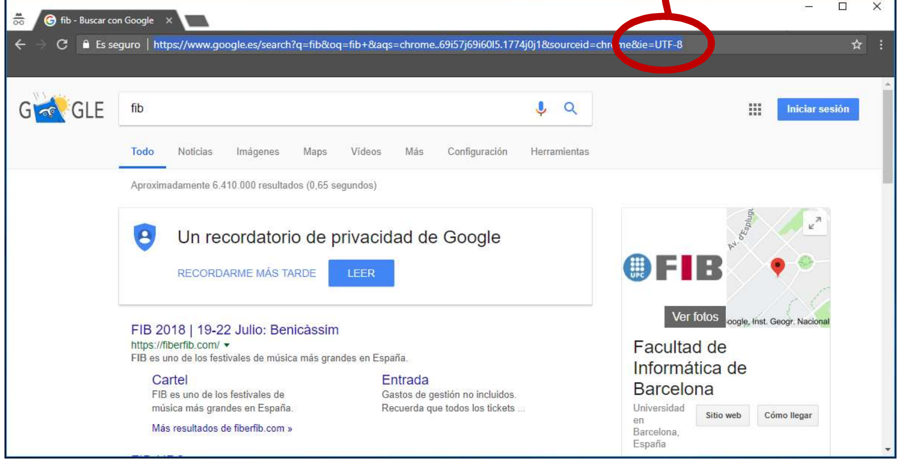
- Several character set encoding
  - ASCII code (American Standard Code for Information Interchange)
    - Single byte using the bottom 7 bits. From 0 to 127
    - The standard for the early HTML
  - ISO-8859-1 code (Latin Alphabet)
    - 1 full byte (256 characters): extension to ASCII
    - The standard from HTML 2.0 to HTML 4.01
    - Problems with some symbols
  - Windows-1252 code (CP-1252): also called ANSI
    - It is a superset of ISO-8859-1 (more printable characters)
    - Used by default on legacy components of Microsoft Windows
  - **UNICODE**



Chars: encoding

- Unicode Standard (created at late 80s by Xerox and Apple)
  - **UTF-8** (from 1 Byte up to 4 bytes, if necessary)
    - It is the dominant encoding
    - It is the standard of HTML5 and websites
    - To support every language, even Klingon, and emojis
  - Several widths
    - UTF-16 (2-4 Bytes)
    - UTF-32 (4 Bytes)

Input Encoding = UTF-8



Chars: encoding issues

- Single Byte versus multiple bytes
  - Fixed-size versus variable-size characters
  - The need to represent character sets that cannot be represented in a single byte (e.g. Japanese, Chinese)
    - MBCSs: MultiByte Character Sets (old and legacy technology)
    - Unicode Standard
- Compatibility issues
  - Unicode-aware programs to manipulate data
    - E.g. fulfilled copy of null-terminated strings (correct process of zero bytes)

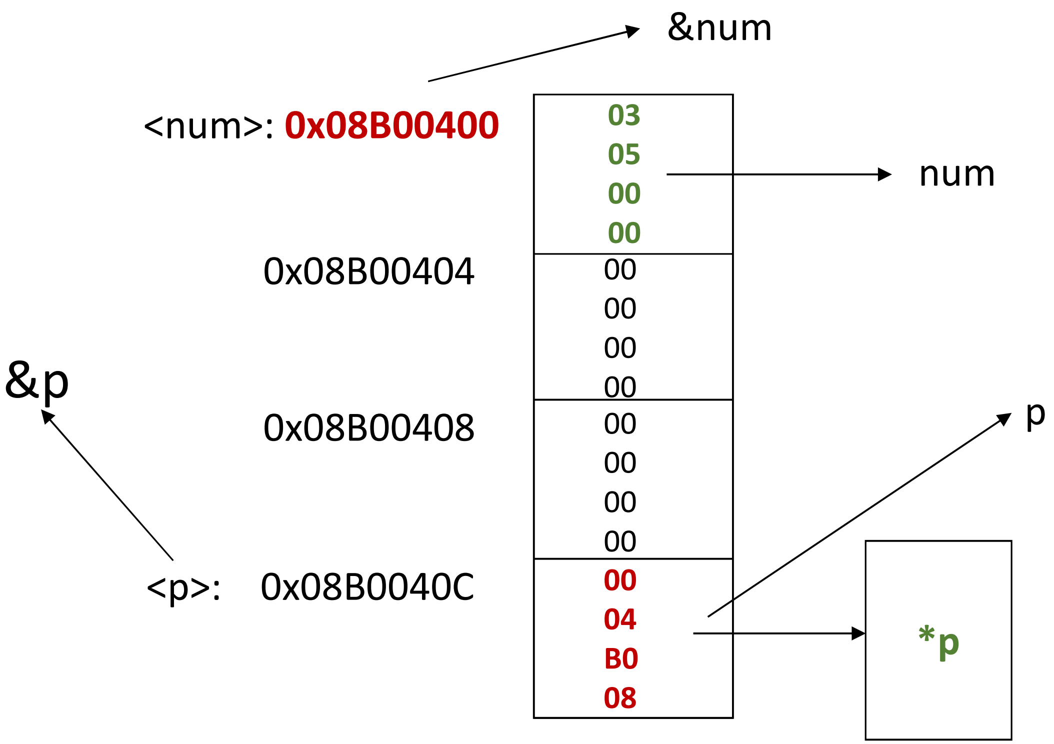
Scalar Data Type: enumeration and boolean

- Enumeration:
  - Ordered list of symbolic names bound to unique values
    - E.g.: colours {red, green, blue}
  - Integer size, by default

- Boolean:
  - True or false
  - Built-in data type in C++ and Python
    - But not in C (integer value: 0 vs 1)
  - Even it only needs one bit, it takes a byte
    - It must be addressable

Scalar Data Type: Pointer

- Value that refers to another value stored elsewhere




```
int num = 0x0503;    //1283
int *p;
p = &num;
```

num = **0x00000503**
&num = **0x08B00400**
p   = **0x08B00400**
&p   = 0x08B0040C
*p   = **0x00000503**

Special Data Type: void and void*

- The void data type is a keyword that refers to a placeholder to a data type to represent no data!!!!!

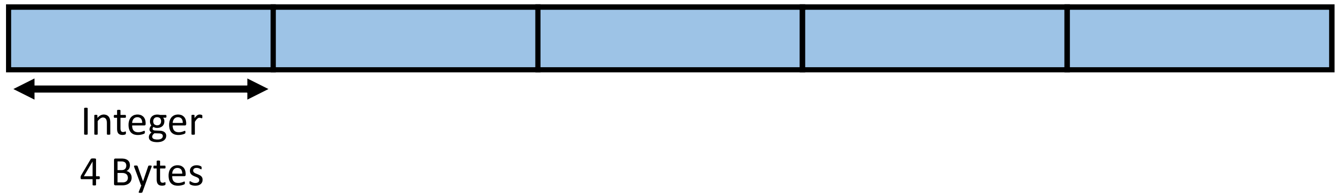
- Different meanings
  - Void function: the function returns nothing
  - Void parameter: no parameter is required

- **Void *** is different…
  - It is a pointer that points to an unespecified data type…that is, anything!!!!
  - Really useful, but take care…

https://msdn.microsoft.com/en-us/library/fxky5d0w.aspx

Aggregate Data Type: arrays

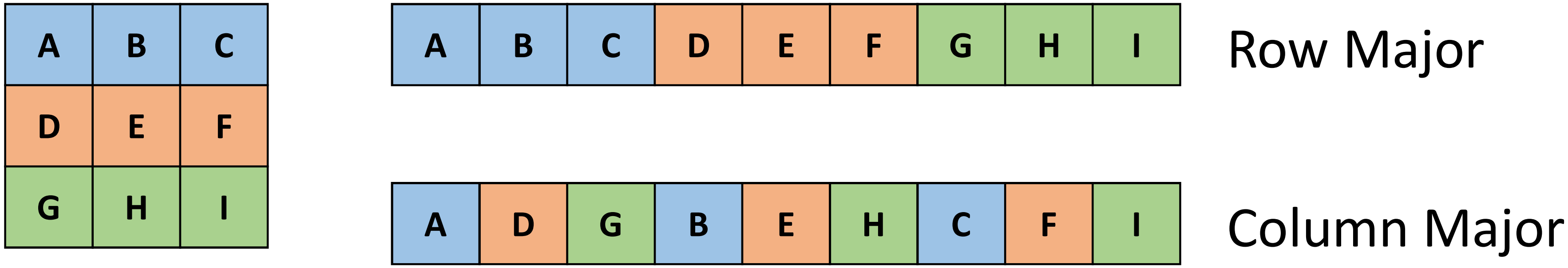
- A collection of items that can be selected/accessed using identifying key/s
  - E.g.: A collection of 5 integers in C/C++: int array[5];



- Implementation complexities
  - Elements of (same vs different) data type/size
  - Indexing keys: integer vs arbitrary values
  - Static vs dynamic array size
  - One vs multiple dimensions

Arrays: tradeoffs

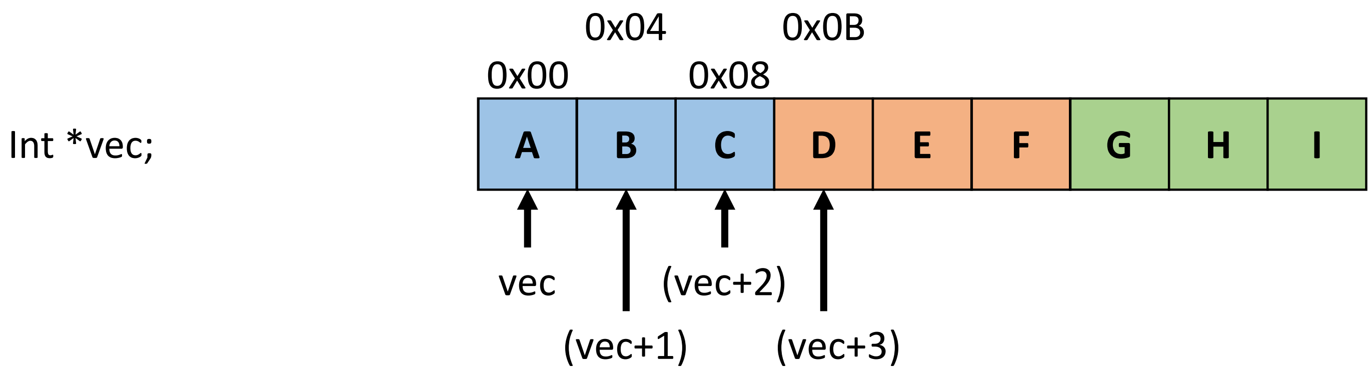
- Memory layout for multidimensional arrays
  - Row-major versus column-major versus depth (for 3D) versus…



  - Programming language based
    - Row major: C, C++, Python
    - Column major: Fortran, MatLab, R
  - Hardware support to boost performance
    - Special registers

Pointer Arithmetics

- Pointers can point to subsequent mem @s based on the data type hold on
  - E.g. difference between **char* (1@mem shift)** and **int* (4@mem shift)**



- Pointer point to a given @ independent of the memory region
  - More details mem @s in future lessons…

Aggregate Data Type: structures

- A record that groups several variables and place them in a particular memory block, accessed by a single pointer or variable name

```
Struct test{
    int enter;
    char caracter;
    float decimal;
}
```

integer (4 Bytes)	char (1 B)	**padding (3 Bytes)**	Float (4 Bytes)

- Padding bytes to align fields in memory aiming at boost performance
  - Processor architecture related (32 bits vs 64 bits)

Aggregate Data Type: unions

- Several values of different type can be accessed in the SAME mem @
    - Only one value at a time
    - Overwrite values

```
Union test{
   int enter;
   char caracter;
   float decimal;
}
```

integer
char
float
(4 Bytes)

- Efficient way to use the same memory location for multiple purpose
- For type-less processing

Other aggregate data types related to OOP

- Object Oriented Programming (OOP): programming paradigm
  - Object is an instance of a class: a combination of variables, functions, and structs
    - E.g.: C++ and Python

- Classes are an evolution of structs
  - Contents (fields) with restricted access

- String is different that an array of chars
  - String is a class-based data type
    - With embedded functions and fields
  - Array of chars is just an array of chars!!!

Bibliography

- Computer Organization and Design (5th Edition)
  - D. Patterson and J. Hennessy
  - http://cataleg.upc.edu/record=b1431482~S1*cat
    - Several chapters introduce different types of data

- Data Type Summary
  - MSDN (Microsoft)
  - https://docs.microsoft.com/en-us/office/vba/language/reference/user-interface-help/data-type-summary
    - Summary list of data types and their respective size in Windows based software

- C++ Language Tutorial
  - http://www.cplusplus.com/doc/tutorial/
    - Summary of basic and compound data types for C++