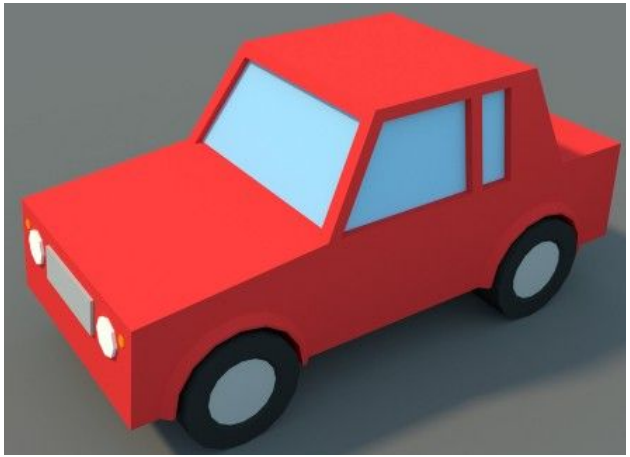
Data Structures

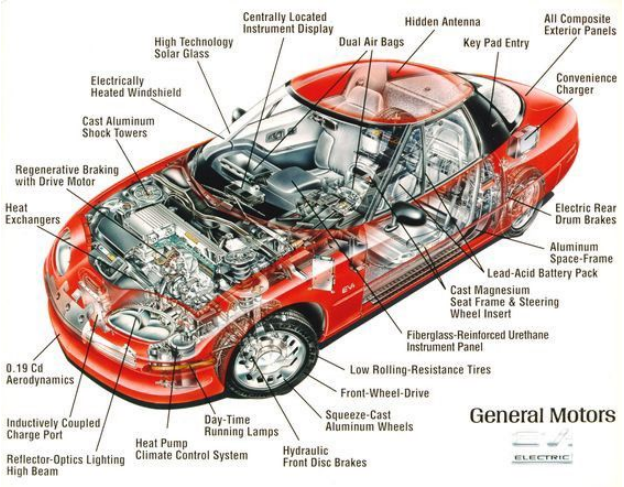
Abstract Data Type

Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

What vs How





Img Src

What vs How

- Do you care how:
  - a TV/Car work? Google really searches and find results? Browser access the internet?
  - C++ computes pow(2.0, 3.2)
  - C++ handles OS to read/write from files using fstream?
- Most of the time, the user care about WHAT not HOW
  - What = Function takes and return
  - How = it is implemented. But
    - Some implementation can be slow (loop to sum 1 to n) or fast (sum = n * n+1 / 2)
    - Some might be buggy or stable (internet explorer vs Firefox)
    - Some might takes more memory (chrome vs Firefox)
  - We can **change** the internal implementation of the class **independently** without affecting the user.
    - User depends on **limited visible** functionalities of specific WHAT details

Data Types

- **Primitive** Data Type
  - E.g. int, float
  - Supported Operations: e.g. x + 2 * y
- **User-Defined** Data Type
  - E.g. Our vector
  - Supported Operations: push_back, find, get(idx)
- What is an **Abstract** Data Type?
  - It is like a user defined data type
  - But we **focus** on the **what**: e.g. push_back
  - But we **don't** care about **how** (not specified yet)
    - is it slow push_back or push_back with capacity trick?
  - There is only ONE what, but MANY how

What is ADT?

- ADTs are a **theoretical** concept. More like logical/mathematical view
  - We specify the **what** part and also potentially the expected **performance**
  - **It is independent** of a programming language and **how** it will be implemented
- Data structures are **concrete**. They are implementing the ADT
  - E.g. providing a push_back functionality with **capacity** enhancement
- The word abstraction?!
  - Abstraction is about **hiding** unwanted details while **showing** most essential in a given context
  - So we show the expected ‘what’ is supported and hide the how
  - Abstraction = High-level
    - Tip: Senior managers have high abstraction skills
    - They focus on the **big picture** and let the **technical** details for the engineers

Why ADT?

- Recall when you learned STL vector or stack, did u care how implemented?
  - Similarly, when you first learned driving, you never care of the inner details
- ADT are acting like an **interface**
  - We as clients: use it based on the agreed provided functionality (interface)
  - The implementer: follow the agreed design (interface)
- In industry
  - You discuss with your team lead the proper interface (provided functionalities / logic)
  - Then implement it

Vector ADT

- get-size()
- get-front()
- get-back()
- get(idx)
- set(idx, val)
- print()
- find(value): Return the **position** of the given value or **None** if not found
  - In C++, the position will be 0-based index. We can return -1 if not found
- push-back(value): Expected to add an element to the end of the vector

C++: Headers for interface separation

```
 5 class Vector {
 6 private:
 7     int *arr { nullptr };
 8     int size { 0 };
 9     int capacity { };
10     void expand_capacity();
11
12 public:
13     Vector(int size);
14     ~Vector();
15
16     int get_size();
17     int get_front();
18     int get_back();
19     int get(int idx);
20     void set(int idx, int val);
21
22     void print();
23     int find(int value);
24     void push_back(int value);
25 };
```

```
74 int Vector::find(int value) {
75     for (int i = 0; i < size; ++i)
76         if (arr[i] == value)
77             return i;
78     return -1;  // -1 for NOT found
79 }
```

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."