Data Structures

Vector

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)

Array limitation

- A C++ built-in array is mainly limited by its initial size
  - Once size is determined, it is fixed
  - This means it is not flexible to insert/remove elements that affects the size
- In practice, we need something that is more memory flexible
  - Such as append (push_back), insert and delete
- In addition, we might want to support some operations. Examples
  - Search for an element, get the minimum element, sort content and so on
  - Definitely also we need to set and get values from the array (e.g. [])
- Let’s create such a dynamic array
  - In C++, we may it vector, similar to STL Vector class

Thinking

- Let’s say we will create a class: Vector (capital V)
- We need to think about our data and operations
- For Data
  - Vector is just a dynamic flexible array
  - It is intuitive to have an array internally, but this time with pointers, e.g. int*
  - We also need to have a size for the current array! ⇒ int size
- Operations
  - Whatever an interesting operation, we just provide a member function to it!
  - E.g. insert function or search function

Let’s code

- Create a class
- Add your data in private section
  - Let’s focus on integers
  - Array size
- Our constructor
  - Just create memory
- Destructor
  - To handle memory leak
  - Sometimes I might skip it, only for simplicity

```
5  class Vector {
6  private:
7      int *arr = nullptr;
8      int size = 0;
9
10 public:
11     Vector(int size) :
12             size(size) {
13         if (size < 0)
14             size = 1;
15         arr = new int[size] { };
16     }
17
18     ~Vector() {
19         delete[] arr;
20         arr = nullptr;
21     }
```

Get and Set

- In arrays, we can set/get data using []
- Let’s provide similar functionalities

```
int get(int idx) {
    assert(0 <= idx && idx < size);
    return arr[idx];
}

void set(int idx, int val) {
    assert(0 <= idx && idx < size);
    arr[idx] = val;
}
```

Operations

- We can add more functions to match our needs
- E.g. print array
- E.g. find index of a value
  - Or -1 if doesn't exist

```
void print() {
    for (int i = 0; i < size; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}

int find(int value) {
    for (int i = 0; i < size; ++i)
        if (arr[i] == value)
            return i;
    return -1;  // -1 for NOT found
}
```

Usage

- We simply create our own object and start to use the functions

```
Vector v(10);
for (int i = 0; i < 10; ++i)
    v.set(i, i);

v.print();
cout<<v.find(5)<<" "<<v.find(55);   // 5 -1
```

Data Structure

- As you see, we are done with our first basic data structure
  - Vector is one of the basic but heavily used data structures
- We introduced data and some operations around it
- In next videos, we see a real need for such a data structure

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."