C++

Subramaniyan

Arrays: - Collection of elements of Same DT
↳ No bound's checking

int A[10]; → ~~Initializing~~ Declaration

int A[10] = {1,2,3,4,5,6,7,8,9,10}
↳ Initialization + Declaration

int A[] = {1,2,3, ... n}

Cin >> A[0];

~~Ar~~ A[11] → out of bounds

Vectors:

→ Container in the C++ Standard Template Library

→ An array that Can grow/Shrink in Size at execution time.

→ Provides similar semantics & syntax as arrays.

→ Very efficient

→ Can provide bound's checking

→ Can use lots of functions like sort, reverse, find & more.

When we create a C++ vector
↳ we create a C++ object
The object can perform operations for us.

<u>Declaring</u>:

#include < vector> → Include vector library
using namespace std;

vector <char> vowels; } Empty vector with no elements
vector <int> test_scores; }
↳ Must include the type of the elements of the vector inside < > angle brackets, since the vector is an obj-oriented <u>template</u> class

∴ vowels → vector or a collection of characters
test_scores → " " of integers.

vector <char> vowels (5);
↳ Constructor initializer syntax

vector <int> test_scores (10);
↳ Automatically set to zero

```
vector <char> vowels {'a', 'e', 'i', 'o', 'u'};

vector <int> test_scores {100, 98, 97, 39, 45};

vector <double> hi_temp (365, 80.0)
```

First Param (size of the vector)

Second Param (Value that will initialize all 365 doubles with 80.0)

Charechteristics:

- Dynamic Size
- Elements are all same type
- Stored Contiguously in memory
- Individual elements can be accessed by their position or index.

- First element is at index 0
- Last element is at index -1

- [ ] - no checking to see if out of bounds
- But provides many useful functions that do bounds check
- Elements initialized to zero
- Very efficient
- Iteration/looping is used to process vectors.

Accessing vector elements - vector syntax

vector_name.at(element_index)

test_scores.at(i)

$\text{test\_scores}\underbrace{.}_{\text{Dotoperator}}\underbrace{\text{at}}_{\text{method-name (type of operation)}}(i)$

Changing Contents of vector elements:

vector_name . at (element index)

```
cin >> test_scores.at(0);
   "
   "
cin >> test_scores.at(4);

test_scores.at(0) = 90 // assignment statement
```

When do A vectors grow as needed;

vector_name.push_back(element)
↳ adds new element to the end of the vector (same type - all elements)

```
vector<int> test_scores{100, 95, 99}; // size 3
test_scores.push_back(80);    // 100, 95, 99, 80
test_scores.push_back(90);    // 100, 95, 99, 80, 90
```

vector will automatically allocate the required space.

Vector will take care of allocating or de-allocating space

What if we are out of bounds in vector?

↳ Many vector methods provide bounds checking

↳ error message / exception is generated.

Example of a 2D-vector:

2D-vector → vector of vectors

```
vector <vector<int>> movie_ratings {
        {1, 2, 3, 4},
        {3, 4, 2, 1},
        {2, 5, 3, 4},
};
```

For a 2D-vector, .at() -method is,

↳ movi_ratings.at(i).at(j)