DAY-3 ARRAYS

C++ provides a data structure, **the array**, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Declaring Arrays

SYNTAX: **type arrayName [arraySize];**

EXAMPLE : double balance[10];

Initializing Arrays

1) INTIALISING UPTO N ELEMENTS IN ARRAY :

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
   int n;
   cout << "Enter the size of the array: ";
   cin >> n;
   vector<int> myVector(n);
   cout << "Enter " << n << " elements for the array:" << endl;
   for (int i = 0; i < n; ++i) {
      cin >> myVector[i];
   }
   cout << "Array elements are: ";
   for (int i = 0; i < n; ++i) {
      cout << myVector[i] << " ";
   }
   // No need to manually delete memory, as vector handles it automatically
   return 0;
}
```

2) **UPTO 5 ELEMENTS** : double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};

USE OF VECTOR IN C++

std::vector in C++:

Dynamic Array:

Purpose: Dynamically manages memory, creating arrays of a size determined at runtime.

Automatic Resizing:

Purpose: Adjusts its size automatically when elements are added or removed.

Iterating Over Elements:

Purpose: Provides convenient iterators for easy traversal of elements.

Efficient Random Access:

Purpose: Allows fast access to elements using random access iterators.

Appending Elements:

Purpose: Easily appends elements to the end of the vector.

Inserting Elements at Specific Positions:

Purpose: Inserts elements at any position within the vector.

Removing Elements:

Purpose: Removes elements from the vector, resizing it automatically.

Sorting Elements:

Purpose: Easily sorts the elements of the vector.

Size Information:

Purpose: Obtains the size and capacity of the vector.

Memory Efficiency:

Purpose: Efficiently manages memory, automatically resizing when needed.

Range-based Algorithms:

Purpose: Uses range-based algorithms provided by the STL.

Custom Data Types:

Purpose: Uses std::vector with custom data types.

SYNTAX IN A CODE :

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   // 1. Creating a Dynamic Array
   vector<int> dynamicArray;
   // 2. Adding Elements to the End
   dynamicArray.push_back(42);
   dynamicArray.push_back(10);
   // 3. Iterating Over Elements
   for (const auto &element : dynamicArray) {
      cout << element << " ";  // Print each element
   }
   cout << endl;
   // 4. Accessing Elements by Index
   int value = dynamicArray[0];
   cout << "First element: " << value << endl;
   // 5. Inserting Element at a Specific Position
   dynamicArray.insert(dynamicArray.begin() + 1, 99);
   // 6. Removing the Last Element
   dynamicArray.pop_back();
   // 7. Sorting Elements
   sort(dynamicArray.begin(), dynamicArray.end());
   // 8. Displaying Size Information
   size_t size = dynamicArray.size();
   size_t capacity = dynamicArray.capacity();
   cout << "Size: " << size << ", Capacity: " << capacity << endl;
   // 9. Memory Efficiency
   dynamicArray.shrink_to_fit(); // Reduce capacity to fit the size
```

```
    // 10. Iterating Over Elements Again
    for (const auto &element : dynamicArray) {
        cout << element << " ";  // Print each element
    }
    cout << endl;
    // 11. Using Custom Data Types
    struct MyStruct { /* ... */ };
    vector<MyStruct> myStructVector;
    return 0;
}
```

SOLVED 10 IMP BASIC PROBLEMS IN LEETCODE

1) https://leetcode.com/problems/max-value-of-equation/

```
class Solution {
public:
  int findMaxValueOfEquation(vector<vector<int>>& points, int k) {
    int ans = INT_MIN;
    priority_queue<pair<int, int>> maxHeap;  // (y - x, x)
    for (const vector<int>& p : points) {
      const int x = p[0];
      const int y = p[1];
      while (!maxHeap.empty() && x - maxHeap.top().second > k)
        maxHeap.pop();
      if (!maxHeap.empty())
        ans = max(ans, x + y + maxHeap.top().first);
      maxHeap.emplace(y - x, x);
    }

    return ans;
  }
};
```

2) https://leetcode.com/problems/best-time-to-buy-and-sell-stock-iii/

```
class Solution {
public:
  int maxProfit(vector<int>& prices) {
    int sellTwo = 0;
    int holdTwo = INT_MIN;
    int sellOne = 0;
    int holdOne = INT_MIN;
```

```
    for (const int price : prices) {
      sellTwo = max(sellTwo, holdTwo + price);
      holdTwo = max(holdTwo, sellOne - price);
      sellOne = max(sellOne, holdOne + price);
      holdOne = max(holdOne, -price);
    }

    return sellTwo;
  }
};
```

3) https://leetcode.com/problems/largest-rectangle-in-histogram/

```
class Solution {
 public:
  int largestRectangleArea(vector<int>& heights) {
    int ans = 0;
    stack<int> stack;

    for (int i = 0; i <= heights.size(); ++i) {
      while (!stack.empty() &&
             (i == heights.size() || heights[stack.top()] > heights[i])) {
        const int h = heights[stack.top()];
        stack.pop();
        const int w = stack.empty() ? i : i - stack.top() - 1;
        ans = max(ans, h * w);
      }
      stack.push(i);
    }

    return ans;
  }
};
```

4) https://leetcode.com/problems/first-missing-positive/

```
class Solution {
 public:
  int firstMissingPositive(vector<int>& nums) {
    const int n = nums.size();
    for (int i = 0; i < n; ++i)
      while (nums[i] > 0 && nums[i] <= n && nums[i] != nums[nums[i] - 1])
        swap(nums[i], nums[nums[i] - 1]);
    for (int i = 0; i < n; ++i)
      if (nums[i] != i + 1)
        return i + 1;
    return n + 1;
  }
};
```

5) https://leetcode.com/problems/subarray-sums-divisible-by-k/

```
class Solution {
 public:
  int subarraysDivByK(vector<int>& nums, int k) {
    int ans = 0;
    int prefix = 0;
    vector<int> count(k);
    count[0] = 1;
    for (const int num : nums) {
      prefix = (prefix + num % k + k) % k;
      ans += count[prefix];
      ++count[prefix];
    }
    return ans;
  }
};
```

6) https://leetcode.com/problems/jump-game/

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int mx = 0;
        for (int i = 0; i < nums.size(); ++i) {
            if (mx < i) {
                return false;
            }
            mx = max(mx, i + nums[i]);
        }
        return true;
    }
};
```

7) https://leetcode.com/problems/invalid-transactions/

```
class Solution {
public:
    vector<string> invalidTransactions(vector<string>& transactions){
        unordered_map<string, vector<tuple<int, string, int>>> d;
        unordered_set<int> idx;
        for (int i = 0; i < transactions.size(); ++i) {
            vector<string> e = split(transactions[i], ',');
            string name = e[0];
            int time = stoi(e[1]);
            int amount = stoi(e[2]);
            string city = e[3];
            d[name].push_back({time, city, i});
            if (amount > 1000) {
                idx.insert(i);
```

```
            }
            for (auto& [t, c, j] : d[name]) {
                if (c != city && abs(time - t) <= 60) {
                    idx.insert(i);
                    idx.insert(j);
                }
            }
        }
        vector<string> ans;
        for (int i : idx) {
            ans.emplace_back(transactions[i]);
        }
        return ans;
    }
    vector<string> split(string& s, char delim) {
        stringstream ss(s);
        string item;
        vector<string> res;
        while (getline(ss, item, delim)) {
            res.emplace_back(item);
        }
        return res;
    }
};
```

8) https://leetcode.com/problems/two-sum/

```
class Solution {
public:
    vector<int> twoSum(vector<int>& nums, int target)
    {
        unordered_map<int, int> m;
        for (int i = 0;; ++i)
        {
            int x = nums[i];
            int y = target - x;
            if (m.count(y))
            {
                return {m[y], i};
            }
            m[x] = i;
        }
    }
};
```

9) https://leetcode.com/problems/best-time-to-buy-and-sell-stock/

```
class Solution
{
public:
```

```
    int maxProfit(vector<int>& prices)
    {
        int ans = 0, mi = prices[0];
        for (int& v : prices)
        {
            ans = max(ans, v - mi);
            mi = min(mi, v);
        }
        return ans;
    }
};
```

10) https://leetcode.com/problems/best-time-to-buy-and-sell-stock-ii/

```
class Solution {
public:
    int maxProfit(vector<int>& prices) {
        int ans = 0;
        for (int i = 1; i < prices.size(); ++i) ans += max(0, prices[i] - prices[i
- 1]);
        return ans;
    }
};
```