🧩 1. Conceito de Array (Vetor)

Um **array** é uma **estrutura de dados linear** que armazena **vários elementos do mesmo tipo** (como int, float, char, etc.) **em posições consecutivas na memória**.

👉 Isso significa que cada elemento pode ser acessado diretamente pelo seu **índice**.

Exemplo simples:

💡 O primeiro elemento sempre está no **índice 0**.

Então, se o vetor tem 5 elementos, os índices vão de 0 a 4.

```
#include <iostream>
using namespace std;

int main() {
   int numeros[5] = {10, 20, 30, 40, 50};

   cout << "Primeiro elemento: " << numeros[0] << endl;
   cout << "Último elemento: " << numeros[4] << endl;

   return 0;
}
```

⚙️ 2. Armazenamento e Acesso aos Dados

Os arrays são armazenados de forma **contígua na memória**, o que permite **acesso direto e rápido** (tempo constante O(1)).

Fórmula de posição na memória:

Endereço do elemento = Endereço inicial + (índice × tamanho do tipo)

Exemplo ilustrativo:

```
// Suponha que "numeros" começa no endereço 1000 e cada int ocupa 4 bytes
// numeros[2] estará em 1000 + (2 * 4) = 1008
```

📦 3. Tipos de Vetores

🔹 Vetor Estático

O **tamanho é fixo** e definido na compilação.

É mais simples, mas não pode crescer nem diminuir.

```
int notas[4]; // tamanho fixo

notas[0] = 7;

notas[1] = 8;

notas[2] = 6;

notas[3] = 10;
```

Vantagens:

- Simples de usar
- Rápido acesso aos dados
- Gerenciamento automático de memória

Desvantagens:

- Tamanho fixo
- Pode desperdiçar memória se o vetor for maior que o necessário

🔹 Vetor Dinâmico

O tamanho **pode ser alterado em tempo de execução**, com **ponteiros** e **alocação dinâmica**.

```
#include <iostream>
using namespace std;

int main() {
    int tamanho;
    cout << "Digite o tamanho do vetor: ";
    cin >> tamanho;

    int* vetor = new int[tamanho]; // alocação dinâmica

    for (int i = 0; i < tamanho; i++) {
        cout << "Digite o valor do elemento " << i << ": ";
        cin >> vetor[i];
    }

    cout << "Valores digitados: ";
    for (int i = 0; i < tamanho; i++) {
        cout << vetor[i] << " ";
    }
```

```
    delete[] vetor; // libera a memória alocada

    return 0;
}
```

Vantagens:

- Pode crescer ou diminuir conforme necessário
- Usa memória de forma mais eficiente

Desvantagens:

- Mais complexo de gerenciar
- Pode causar vazamento de memória se delete não for usado corretamente

⚖️ 4. Vantagens e Desvantagens Gerais

Vantagens	**Desvantagens**
Acesso rápido O(1)	Inserções e remoções no meio são lentas O(n)
Simples de implementar	Tamanho fixo (vetor estático)
Eficiente em memória (estático)	Pode desperdiçar memória (dinâmico)

💭 5. Reflexão (do PDF)

O texto final te leva a pensar:

Será que existe uma estrutura de dados que **permite inserções e remoções mais eficientes** no meio, sem precisar mover tantos elementos?

✅ A resposta é **sim!**

Estruturas como **Listas Encadeadas (Linked Lists)** resolvem esse problema, pois os elementos não precisam estar lado a lado na memória — mas isso é o próximo passo da disciplina.