FOR

- Laço de repetição que permite executar um bloco de código várias vezes.
- Ideal quando você sabe quantas vezes quer repetir uma ação.
- **Exemplo Básico:** Este exemplo imprime os números de 0 a 4.

```
for(int i = 0; i < 5; i++){
    cout << i << endl;
}
```

- **Exemplo Genérico:**

```
for(inicialização; condição;
incremento i++/decremento i--){
    // bloco de código
}
```

- **inicialização:** define uma variável que controla o loop, geralmente 0 ou 1, se não for decrescente;
- **condição:** verifica se o loop deve continuar, os comandos são exibidos enquanto a condição for V;
- **incremento/decremento:** ajusta a variável após cada iteração, quantas un ela irá aumentar ou diminuir;

WHILE

- O while é outro tipo de laço de repetição, mas continua executando o bloco de código enquanto a condição for verdadeira.
- Ideal quando você não sabe exatamente quantas vezes quer repetir algo.
- **Exemplo Básico:** Este exemplo também imprime os números de 0 a 4.

```
int i = 0;
while(i < 5) {
    cout << i << endl;
    i++;
}
```

- **Exemplo Genérico:**

```
while(condição) {
    // bloco de código
}
```

- **condição:** determina se o loop continua ou não;
- **inicialização:** deve se inicializar a variável de iteração na linha de declaração;
- **incremento/decremento:** deve ser adicionado no final do bloco de comandos dentro do while;

DO… WHILE…

- O do while é parecido com o while, mas a diferença é que o bloco de código é executado pelo menos uma vez antes de a condição ser verificada.
- **Exemplo Básico:** Este exemplo imprime os números de 0 a 4.

```
int i = 0;
do {
    cout << i << endl;
    i++;
} while(i < 5);
```

- **Exemplo Genérico:**

```
do {
    // bloco de código
} while(condição);
```

- **condição:** o loop continua enquanto for verdadeira, mas o código dentro do do é executado pelo menos uma vez;
- **inicialização:** deve se inicializar a variável de iteração na linha de declaração;
- **incremento/decremento:** deve ser adicionado no final do bloco de comandos dentro do while;

ARRAY

- É uma coleção de elementos do mesmo tipo, armazenados em posições consecutivas na memória.
- É útil para armazenar múltiplos valores relacionados.
- "Imagine um conjunto de caixas numeradas em sequência, onde cada caixa só pode guardar um tipo específico de objeto, como bolas. Se você tem 10 bolas, pode colocar uma em cada caixa numerada de 0 a 9. Agora, sempre que precisar de uma bola específica, você sabe exatamente em qual caixa procurar. Esse conjunto de caixas é o array, e cada caixa é um "elemento" do array."
- **Exemplo Básico:** Neste exemplo, criamos um array de inteiros com 5 elementos.

```
int numeros[5] = {10, 20, 30, 40,
50};
cout << numeros[2]; // Imprime 30
```

- **Exemplo Genérico:**

```
tipo nome_do_array[tamanho];
```

- **tipo:** o tipo de dado que o array vai armazenar;
- **nome_do_array:** o nome do array;
- **tamanho:** o número de elementos que o array pode conter;

VETOR

- Tecnicamente, em C++, vetor e array são sinônimos, mas muitas vezes o termo "vetor" refere-se à classe vector, que é mais flexível e faz parte da biblioteca padrão.
- Ao contrário de arrays fixos, vetores podem redimensionar automaticamente.
- **Exemplo Básico:** Neste exemplo, usamos um vetor da biblioteca padrão.

```
#include <vector>
vector<int> numeros = {10, 20, 30,
40, 50};
cout << numeros[2]; // Imprime 30
```

- **Exemplo Genérico:**

```
vector<tipo> nome_do_vetor;
```

- **tipo:** tipo de dado que o vetor vai armazenar.
- **nome_do_vetor:** o nome do vetor.

STRUCT

- Permite agrupar diferentes tipos de dados sob um único nome.
- É útil para organizar informações relacionadas.
- **Exemplo Básico:** Este exemplo cria uma estrutura para armazenar dados de uma pessoa.

```
struct Pessoa {
    string nome;
    int idade;
    float altura;
};
Pessoa p1;
p1.nome = "João";
p1.idade = 30;
p1.altura = 1.75;
cout << p1.nome << " tem " <<
p1.idade << " anos e mede " <<
p1.altura << " metros." << endl;
```

- **Exemplo Genérico:**

```
struct NomeDaStruct {
    tipo campo1;
    tipo campo2;
    // outros campos
};
```

- **NomeDaStruct:** nome da estrutura.
- **campo1, campo2, ...:** variáveis que compõem a estrutura, cada uma com seu tipo.