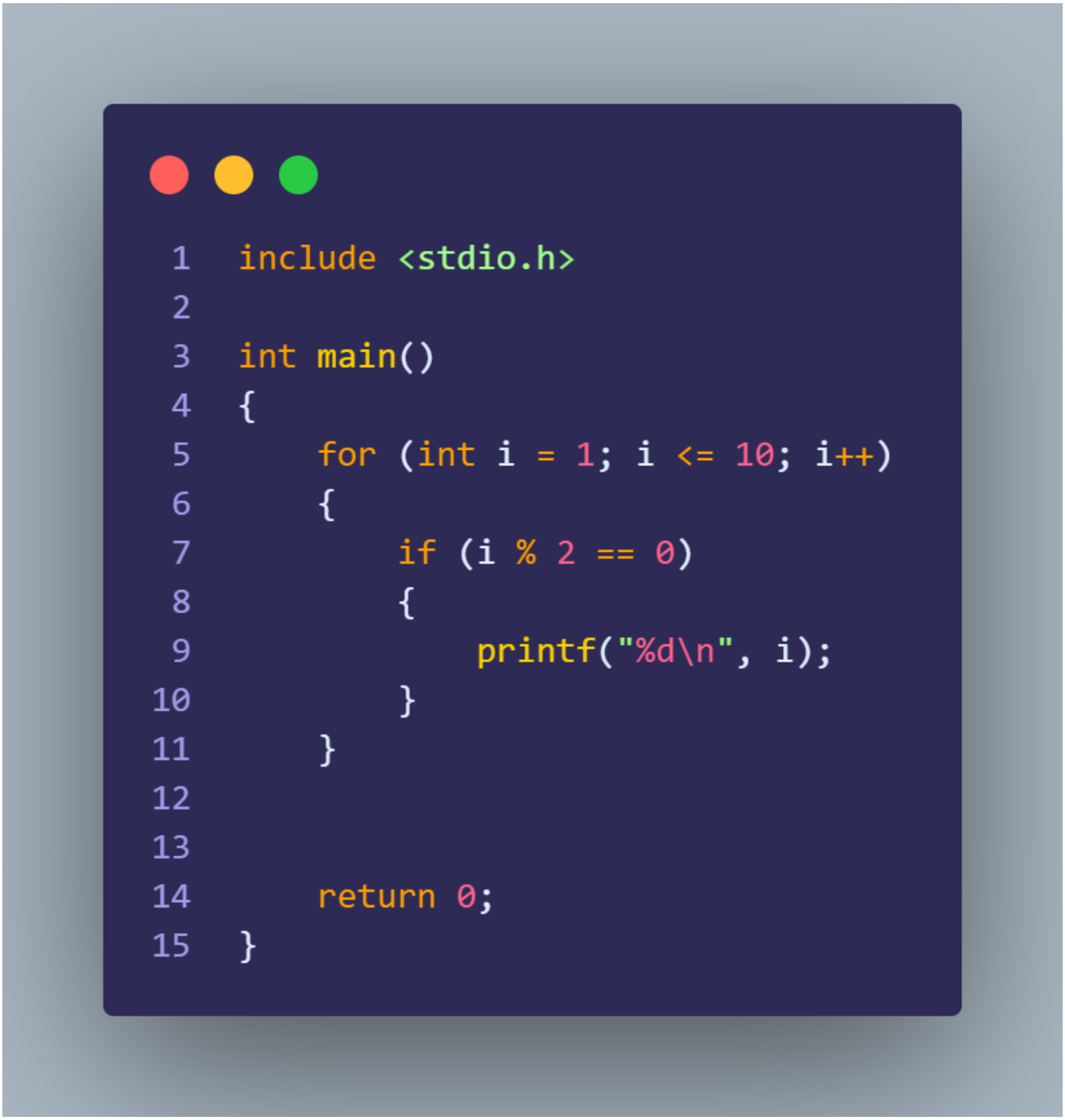
Estruturas de Repetição (Loops)



Nesta semana, você aprenderá como usar estruturas de repetição para executar blocos de código várias vezes em C. Isso é útil para automatizar tarefas repetitivas, como somar uma sequência de números ou pedir informações ao usuário até que uma condição seja satisfeita.

Introdução aos Laços de Repetição

Os laços de repetição, também chamados de loops, são usados quando queremos repetir um trecho de código várias vezes. Em C, os dois tipos principais de laços são o **for** e o **while**.

Vamos ver em detalhes como cada um deles funciona.

Laços com Quantidade Definida de Repetições (for)

O laço `for` é usado quando sabemos quantas vezes o bloco de código deve ser repetido. Ele segue a seguinte estrutura:

```
for (inicialização; condição; incremento/decremento) {
    // bloco de código a ser repetido
}
```

Exemplo Prático: Somar Números de 1 a 100

Aqui está um exemplo de um programa que soma todos os números de 1 a 100 usando um laço `for`:

```
#include <stdio.h>

int main() {
    int soma = 0;

    for (int i = 1; i <= 100; i++) {
        // Cria-se uma variável i que é inicializa em 1
        // O loop será realizado enquanto i for menor ou igual a 100, nesse
caso até ele chegar a 100, no 101 o loop será quebrado
        // O loop será incrementado (i++) enquanto a condição for verdadeira

        /*
        Primeira execução
            i = 1
            1 <= 100 (true)
            soma += 1 (soma = 1)
            i += 1 (i = 2)

        Segunda execução
            i = 2
            2 <= 100 (true)
            soma += 2 (soma = 1 + 2 = 3)
            i += 1 (i = 3)

        ...

        Nonagésima nona execução
            i = 99
            99 <= 100 (true)
            soma += 99 (soma = 1 + 2 + 3 + ... + 98 + 99 = 4950)
            i += 1 (i = 100)

        Centésima execução
            i = 100
            100 <= 100 (true, apesar de 100 não ser menor do que 100 ele é
igual a 100)
            soma += 100 (soma = 1 + 2 + 3 + ... + 99 + 100 = 5050)
```

```
            i += 1 (i = 101)

        Centésima primeira execução
            i = 101
            101 <= 100 (false, opa! agora a condição que roda o loop é falsa,
então, o corpo do for não será executado!)
            Fim do Loop
        */

        soma += i;  // Adiciona o valor de i à soma
    }

    printf("A soma dos números de 1 a 100 é: %d\n", soma);

    return 0;
}
```

Exemplo Prático: Tabuada

Podemos também usar o laço `for` para gerar a tabuada de um número:

```c
#include <stdio.h>

int main() {
    int numero;

    printf("Digite um numero para ver sua tabuada: ");
    scanf("%d", &numero);

    for (int i = 1; i <= 10; i++) {
        printf("%d x %d = %d\n", numero, i, numero * i);
    }

    return 0;
}
```

Neste exemplo, o programa pede um número ao usuário e imprime a tabuada desse número de 1 a 10.

Então:

```
Digite um numero: 5
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
...
5 x 9 = 45
5 x 10 = 50
```

Na condição do loop, apesar de usarmos apenas o operador menor ou igual (<=) qualquer outro operador apresentado na aula passada poderia ser utilizado >=, !=, etc.

Laços com Quantidade Indefinida de Repetições (while)

O laço `while` é utilizado quando não sabemos quantas vezes o código deve ser repetido, e queremos repetir o bloco até que uma condição seja satisfeita.

A estrutura básica de um laço `while` é:

```
while (condição){
    // bloco de código a ser repetido
}
```

Exemplo Prático: Repetição até que uma Condição seja Satisfeita

Um exemplo comum é pedir ao usuário que insira uma senha até que ele digite a senha correta:

```c
#include <stdio.h>

int main() {
    int senha = 1234; // senha pré-definida
    int tentativa;

    printf("Digite a senha: ");
    scanf("%d", &tentativa);

    while (tentativa != senha) {
        printf("Senha incorreta. Tente novamente: ");
        scanf("%d", &tentativa);
    }

    printf("Senha correta! Acesso concedido.\n");

    return 0;
}
```

Neste caso, o loop continuará pedindo a senha ao usuário até que ele digite a senha correta (1234).

Laços com Quantidade Indefinida de Repetições (do while)

O loop `do while` é semelhante ao `while`, mas a diferença principal é que ele garante que o bloco de código seja executado pelo menos uma vez antes de verificar a condição. Veja o exemplo:

```c
#include <stdio.h>

int main() {
```

```
    char resposta;

    do {
        printf("Você gostaria de continuar? (s/n): ");
        scanf(" %c", &resposta);
    } while (resposta == 's' || resposta == 'S');

    printf("Programa encerrado!\n");

    return 0;
}
```

Obrigatoriamente, o usuário deve responder se quer continuar ou não ao menos uma vez. Se estivéssemos em um loop `while`, a leitura deveria ser feito fora do loop para então verificar a condicional.

Diferença entre `for`, `while` e `do-while`

Embora `for`, `while` e `do-while` possam ser usados para repetir blocos de código, cada um tem seu contexto específico de uso:

- `for`: Utilize este laço quando souber o número exato de repetições que deseja realizar. Por exemplo, ao somar números de 1 a 100 ou gerar uma tabuada, o `for` é a escolha ideal.
- `while`: Este laço é apropriado quando a quantidade de repetições não é conhecida de antemão, mas você tem uma condição que determina quando o loop deve parar. Um bom exemplo é pedir ao usuário uma senha até que a correta seja inserida.
- `do-while`: Semelhante ao `while`, mas com uma diferença importante: a condição é verificada após a execução do bloco de código. Isso garante que o código dentro do `do-while` seja executado pelo menos uma vez, independentemente da condição inicial.

Quando Usar Cada Um?

- `for`: Quando a quantidade de repetições é conhecida antes da execução.
- `while`: Quando a repetição depende de uma condição que pode variar durante a execução do programa.
- `do-while`: Quando você deseja garantir que o bloco de código seja executado pelo menos uma vez, mesmo que a condição de parada seja falsa logo no início.

Conclusão

Agora que você entende como os loops `for`, `while` e `do-while` funcionam, pode aplicá-los em diversas situações para automatizar tarefas repetitivas em seus programas. Experimente usar os três tipos de laços em diferentes cenários e veja qual é o mais eficiente para cada caso!

Na próxima aula, praticaremos estruturas condicionais e de repetição por meio de exercícios.