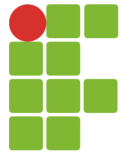
Estruturas de Repetição - Parte 2

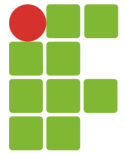
Algoritmos e Programação de Computadores – ABI/LFI/TAI

Prof. Daniel Saad Nogueira Nunes

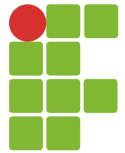
IFB – Instituto Federal de Brasília,
Campus Taguatinga



Sumário

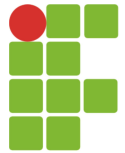


Sumário

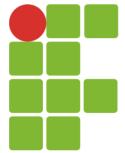


Introdução

- Vimos na aula anterior as estruturas de repetição suportadas pela linguagem C.
- Nesta aula veremos uma série de exemplos em como aplicar as estruturas vistas para resolver diversos problemas.
- Usaremos diferentes padrões de resolução de problemas, os quais são úteis conhecer na hora de resolver problemas novos, pois os mesmos padrões poderão se aplicar.
- Também veremos os comandos de interrupção de fluxo.
- Além disso, abordaremos a questão de como escrever laços infinitos.

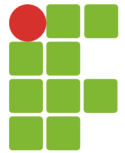


Sumário



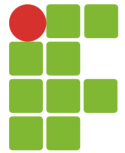
Laços Infinitos

- Nem todo o programa precisa parar de executar.
- Temos diversos exemplos de sistemas que não devem parar de executar, a menos que um comando explícito seja dado.
- Exemplo: Sistema Operacional.
- Podemos escrever laços que nunca param de executar, para que efetue-se um processamento infinito.
- Basta que a condição de parada nunca seja falsa!



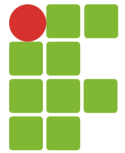
Laços Infinitos: While

```c
#include <stdio.h>

int main(void) {
    int i = 1;
    while (1) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```



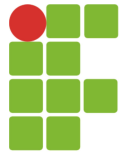
Laços Infinitos: For

```c
#include <stdio.h>

int main(void) {
    int i;
    for (i = 1;; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

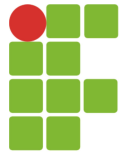


Laços Infinitos: For

```c
#include <stdio.h>

int main(void) {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (1);
    return 0;
}
```

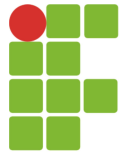


Sumário

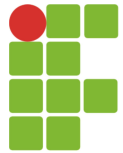


Interrupção

- Existem dois comandos que interrompem o fluxo de execução do laço, mas de maneira diferente:
  - `break` e `continue`.
- Examinaremos cada um deles agora.

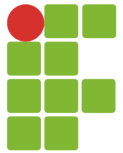


Sumário



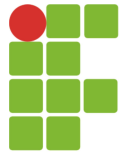
Interrupção: Break

- O comando `break` interrompe o fluxo do laço de repetição, fazendo com que o fluxo do programa continue exatamente após o laço.
- Útil para economizar computação, uma vez que uma determinada condição seja atingida.



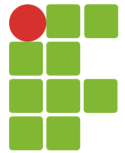
Interrupção: Break

- Por exemplo: vamos tomar o problema de calcular a **menor** potência de dois que é **maior ou igual** do que um determinado inteiro x, lido do teclado. Estamos assumindo que x é sempre positivo.
- Se $x = 5$ a resposta seria 8.
- Se $x = 16$ a resposta é 16.
- Se $x = 600$, a resposta é 1024



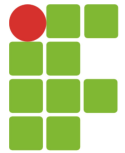
Interrupção: Break

- A ideia aqui é abortar o laço assim que a potência seja encontrada.
- Iremos manter uma variável `pot` que iniciará de 1, a primeira potência inteira de dois, e computará a próxima potência de dois a cada iteração.
- Assim que `pot` seja maior ou igual a x, o laço será abortado.



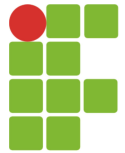
Interrupção: Break

```c
#include <stdio.h>

int main(void) {
    int pot, x;
    printf("Digite um inteiro positivo: ");
    scanf("%d", &x);
    pot = 1;
    while (1) {
        if(pot>=x)
            break;
        pot *= 2;
    }
    printf("Resposta: %d\n",pot);
    return 0;
}
```



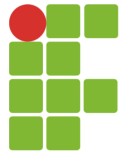
Interrupção: Break

- Equivalentemente, podemos inserir a condição do `break` no laço de repetição de modo a eliminá-lo.
- Em códigos mais complexos, quando temos múltiplas condições de parada, o uso do comando `break` pode simplificar a solução.



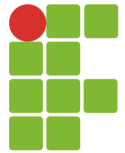
Interrupção: Break

```c
#include <stdio.h>

int main(void) {
    int pot, x;
    printf("Digite um inteiro positivo: ");
    scanf("%d", &x);
    pot = 1;
    while (pot < x)
        pot *= 2;
    printf("Resposta: %d\n", pot);
    return 0;
}
```

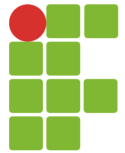


Interrupção: Break

```c
#include <stdio.h>

int main(void) {
    int pot, x;
    printf("Digite um inteiro positivo: ");
    scanf("%d", &x);
    for (pot = 1; pot < x; pot *= 2)
        ;
    printf("Resposta: %d\n", pot);
    return 0;
}
```

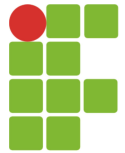


Sumário



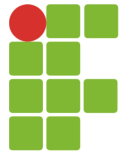
Interrupção: Continue

- O comando `continue`, em vez de sair do laço de repetição como o `break`, ignora restante do bloco de comandos que está abaixo dele e vai direto para o teste da condição.
- Em termos práticos: pula para a próxima iteração do laço, caso haja uma próxima iteração.



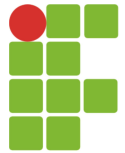
Interrupção: Continue

- Por exemplo: vamos tomar o problema de imprimir todos os números de 1 a 10, exceto o 5.



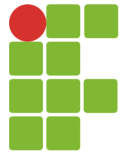
Interrupção: Break

```c
#include <stdio.h>

int main(void) {
    int i;
    for(i=1;i<=10;i++){
        if(i==5){
            continue;
        }
        printf("%d ",i);
    }
    printf("\n");
    return 0;
}
```



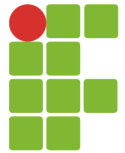
Interrupção: Continue

- Por exemplo: vamos tomar o problema de imprimir a área de todos os círculos com raio entre 1 e 10.
- Contudo, caso o raio seja múltiplo de 4 a área não deve ser impressa.

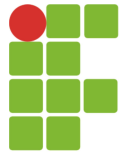


Interrupção: Break

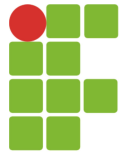
```c
#include <stdio.h>

int main(void) {
    int r;
    const double pi = 3.1415;

    for (r = 1; r <= 10; r++) {
        if (r % 4 == 0)
            continue;
        double area = pi * r * r;
        printf("Área do círculo de raio %d = %.2f\n", r, area);
    }
    return 0;
}
```



Sumário

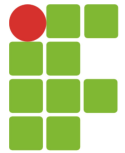


Sumário



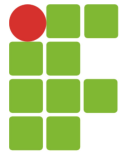
Menu

- Suponha que queiramos construir uma calculadora com as quatro operações básicas.
- A ideia é apresentar um menu pro usuário com as opções de operação e a opção de sair.
- O usuário então escolhe a opção e, caso ela seja válida e não seja a de encerrar o programa, digita os dois números a serem operados, os quais devem ser lidos pelo programa e operados por ele.
- O programa então volta a apresentar as opções para o usuário e tudo se repete, até que ele opte por digitar a opção que finaliza o programa.



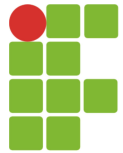
Menu

- Neste caso, é mais natural a escolha da estrutura `do while`, já que o menu deve ser apresentado de qualquer forma, isto é, pelo menos uma iteração do laço deve ser executada.



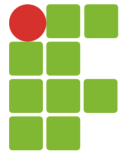
Menu

```
1  #include <stdio.h>
2
3  int main(void) {
4      double op1, op2;
5      int opcao;
6      do {
7          printf("1. Soma\n");
8          printf("2. Subtração\n");
9          printf("3. Multiplicação\n");
10         printf("4. Divisão\n");
11         printf("5. Sair\n");
12         printf("Digite uma das opções (1 a 5): ");
13         scanf("%d", &opcao);
14         if (opcao >= 1 && opcao <= 4) {
15             printf("Digite o primeiro número: ");
16             scanf("%lf", &op1);
17             printf("Digite o segundo número: ");
18             scanf("%lf", &op2);
19         }
```

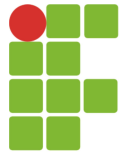


Menu

```
20          if (opcao == 1)
21              printf("%.2f + %.2f = %.2f\n\n", op1, op2, op1 + op2);
22          else if (opcao == 2)
23              printf("%.2f - %.2f = %.2f\n\n", op1, op2, op1 - op2);
24          else if (opcao == 3)
25              printf("%.2f * %.2f = %.2f\n\n", op1, op2, op1 * op2);
26          else if (opcao == 4)
27              printf("%.2f / %.2f = %.2f\n\n", op1, op2, op1 / op2);
28          else if (opcao != 5)
29              printf("Opção inválida, digite números de 1 a 5\n\n");
30      } while (opcao != 5);
31      return 0;
32  }
```

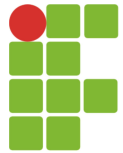


Sumário



Números Primos: Verificação

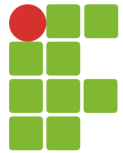
- Um número primo é um número natural que possui **exatamente** dois divisores naturais.
- O número 1 não é primo, pois só possui apenas um divisor: ele mesmo.
- Em outras palavras: se o número é maior que um ele é primo se ele é divisível apenas por ele mesmo e por 1.
- Como criar um programa que verifica se um número é primo?



Números Primos: Verificação

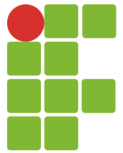
Estratégia 1

- Dado um número x, podemos dividi-lo por todos os números no intervalo $[2, x-1]$.
- Caso ele seja divisível por algum número, então x não pode ser primo.
- Caso x seja maior que 1 e não divisível pelos números entre $[2, x-1]$, então x é primo.
- Exemplo: para o número 5, testaríamos se ele é divisível pelos números 2, 3 e 4.



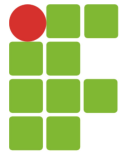
Números Primos: Verificação

```c
#include <stdio.h>

int main(void) {
    int x, i;
    int primo;
    printf("Digite um número natural: ");
    scanf("%d", &x);
    primo = x != 1 ? 1 : 0;
    for (i = 2; i < x; i++) {
        if (x % i == 0)
            primo = 0;
    }
    if (primo)
        printf("%d é primo.\n",x);
    else
        printf("%d não é primo.\n",x);
    return 0;
}
```



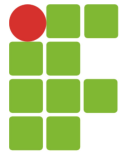
Números Primos: Verificação

- Note que estamos fazendo o uso de uma **variável indicadora** `primo` que indica se o número é ou não é primo.
- Inicialmente ela é verdadeira se o número é maior que 1 e falsa, caso contrário.
- Durante o laço, caso encontremos um número que divida x, o valor falso é atribuído à ela.
- Ao final do processamento, temos que ela é verdadeira se x é primo e falsa quando x não é primo.



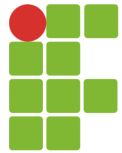
Números Primos: Verificação

- Podemos fazer uma pequena otimização: assim que detectamos que o número não é primo, não há razão para continuar o processamento.
- Utilizamos o valor da variável `primo` como critério de parada!
- Para $x = 9$, testaremos os divisores 2 e 3 apenas, uma vez que 9 é divisível por 3. Não é necessário testar todos os divisores.



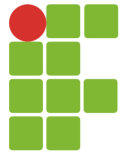
Números Primos: Verificação

```c
#include <stdio.h>

int main(void) {
    int x, i;
    int primo;
    printf("Digite um número natural: ");
    scanf("%d", &x);
    primo = x != 1 ? 1 : 0;
    for (i = 2; i < x && primo; i++) {
        if (x % i == 0)
            primo = 0;
    }
    if (primo)
        printf("%d é primo.\n", x);
    else
        printf("%d não é primo.\n", x);
    return 0;
}
```



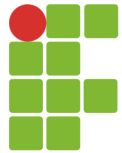
Números Primos: Verificação

```c
#include <stdio.h>

int main(void) {
    int x, i;
    int primo;
    printf("Digite um número natural: ");
    scanf("%d", &x);
    primo = x != 1 ? 1 : 0;
    for (i = 2; i < x; i++) {
        if (x % i == 0){
            primo = 0;
            break;
        }
    }
    if (primo)
        printf("%d é primo.\n", x);
    else
        printf("%d não é primo.\n", x);
    return 0;
}
```



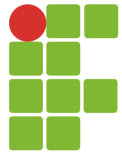
Números Primos: Verificação

- A utilização de uma **variável indicadora** é muito útil em algumas soluções.
- Ela indica se temos (1) ou não temos (0) uma determinada propriedade.
- Pode ser usada para agilizar as computações quando empregada como critério de parada.



Números Primos: Verificação

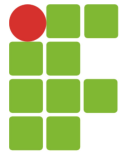
- Os programas apresentados funcionam corretamente.
- Contudo, podemos implementar um código mais eficiente.
- Se um número x não é primo, então ele pode ser escrito como um produto de dois números p e q, com $1 \leq p < x$ e $1 \leq q \leq x$.
- $x = p \cdot q$.
- Observação: p e q não podem ser, **simultaneamente**, maior que $\sqrt{x}$.
- Conclusão: só precisamos testar os divisores até $\sqrt{x}$.



Números Primos: Verificação

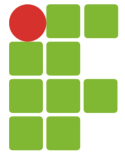
Estratégia 2

- Dado um número x, podemos dividi-lo por todos os números no intervalo $[2, \sqrt{x}]$.
- Caso ele seja divisível por algum número, então x não pode ser primo.
- Caso x seja maior que 1 e não divisível pelos números entre $[2, \sqrt{x}]$, então x é primo.
- Exemplo: para o número 17, testaríamos se ele é divisível pelos números 2, 3 e 4, haja vista que $\sqrt{17} \approx 4.12$.

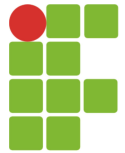


Números Primos: Verificação

```c
#include <math.h>
#include <stdio.h>
int main(void) {
    int x, i;
    int primo;
    printf("Digite um número natural: ");
    scanf("%d", &x);
    primo = x != 1 ? 1 : 0;
    for (i = 2; i <= sqrt(x) && primo; i++) {
        if (x % i == 0)
            primo = 0;
    }
    if (primo)
        printf("%d é primo.\n", x);
    else
        printf("%d não é primo.\n", x);
    return 0;
}
```



Sumário

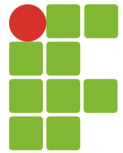


Fatorial

- O fatorial de um número inteiro $n \geq 0$ é definido da seguinte forma:

$$n! = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ n \cdot (n-1) \cdot \ldots \cdot 1, n > 1 \end{cases}$$

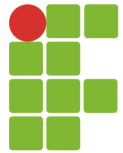
- $0! = 1$.
- $1! = 1$
- $5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$.
- Vamos escrever um programa que calcule $n!$, dado um n lido do usuário.



Fatorial

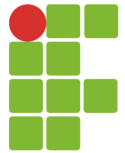
Estratégia

- Uma estratégia para resolver este problema é utilizar uma **variável acumuladora** `fat`.
- Esta variável armazena os valores intermediários da computação do fatorial a cada iteração.
- Ao final da computação, a variável conterá o resultado definitivo.
- Na i-ésima iteração, a variável `fat` conterá o resultado de $i!$.



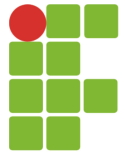
Fatorial

```c
#include <stdio.h>

int main(void) {
    int i;
    int fat;
    int n;
    printf("Digite o valor de n: ");
    scanf("%d", &n);
    for (i = 2, fat = 1; i <= n; i++) {
        fat *=  i;
    }
    printf("%d! = %d\n",n,fat);
    return 0;
}
```

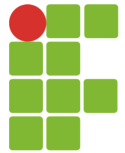


Fatorial

- A utilização de uma **variável acumuladora** está presente nas soluções de muitos problemas.
- Podemos empregá-la sempre que o resultado de uma iteração depende imediatamente da anterior.



Sumário

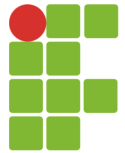


Fibonacci

- O n-ésimo item da sequência de Fibonacci pode ser definido através da seguinte equação:

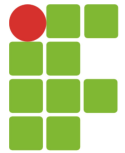
$$F(n) = \begin{cases} 1, & n = 0 \\ 1, & n = 1 \\ F(n-1) + F(n-2), & n > 1 \end{cases} \quad (1)$$

- Assim, a sequência é constituída dos números: $1, 1, 2, 3, 5, 8, 13, \ldots$
- Cada número é igual a soma dos dois anteriores, com exceção dos dois primeiros, que valem 1.



Fibonacci

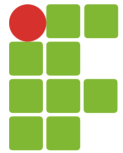
- Problema: dado um inteiro $n \geq 0$, computar $F(n)$.



Fibonacci

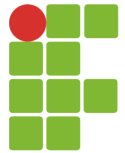
Estratégia

- Podemos resolver este problema mantendo duas variáveis acumuladoras, `fib_1` e `fib_2`.
- A cada iteração estas variáveis são atualizadas para armazenar o último e o penúltimo número de Fibonacci vistos.
- Com a posse delas, é possível calcular o próximo número de Fibonacci.

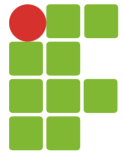


Fibonacci

```c
#include <stdio.h>

int main(void) {
    int i, n;
    int fib_1, fib_2;
    int resposta = 1;
    printf("Digite o valor de n: ");
    scanf("%d", &n);
    for (i = 2, fib_1 = 1, fib_2 = 1; i <= n; i++) {
        resposta = fib_1 + fib_2;
        int aux = resposta;
        fib_2 = fib_1;
        fib_1 = aux;
    }
    printf("F(%d) = %d.\n", n, resposta);
    return 0;
}
```

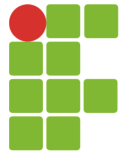


Sumário



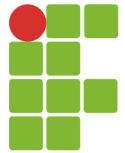
Binário para Decimal

- Problema: dado um número inteiro, contendo apenas dígitos 0 e 1, representando um número binário, convertê-lo para decimal.
- Exemplo: $11001 = 25$.



Binário para Decimal

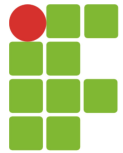
- Como extrair cada dígito?
- Podemos dividir pela potência de 10 adequada e tirar o resto por 10:
  - **1**956: `1956/1000 % 10 == 1`.
  - 1**9**56: `1956/100 % 10 == 9`.
  - 19**5**6: `1956/10 % 10 == 5`.
  - 195**6**: `1956/1 % 10 == 6`.



Binário para Decimal

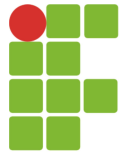
Estratégia

- Podemos utilizar duas variáveis acumuladoras:
  - `pot_2`: armazena o valor da próxima potência de dois.
  - `soma`: armazena o valor da soma de todas as potências de dois cujo bit 1 está ligado.
- A cada iteração `soma` recebe o valor do bit vezes o valor `pot_2`.

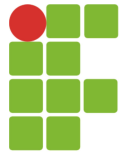


Binário para Decimal

```c
#include <stdio.h>

int main(void) {
    int n;
    int n_original;
    int pot_2 = 1, soma = 0;
    printf("Digite um número em binário: ");
    scanf("%d", &n);
    n_original = n;
    do {
        int bit = n % 10; // obtém o valor do digito menos significativo
        soma += bit * pot_2;
        n /= 10; // divide o número por 10 a cada iteração
        pot_2 *= 2; // obtém a próxima potência de dois
    } while (n > 0);
    printf("%d = %d\n",n_original,soma);
    return 0;
}
```

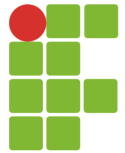


Sumário



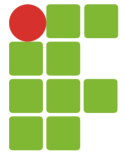
Laços Aninhados

- Em muitos casos, para resolver um problema, é necessário empregar uma solução que envolve laços de repetição dentro de laços de repetição.
- **Laços Aninhados**.



Laços Aninhados

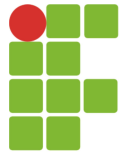
- Problema: imprimir todos os pares de números, que devem estar entre 1 e 100, que somam um inteiro c, informado pelo usuário.



Laços Aninhados

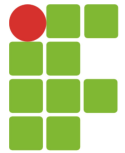
Estratégia

- A ideia é utilizar dois laços aninhados.
- O primeiro laço fixa um valor `i`, enquanto o segundo, interno, varia o valor `j` de 1 a 100.
- Toda vez que o laço interno finaliza, incrementamos o valor de `i` para que se possa reiniciar o processo com j novamente variando de 1 a 100.



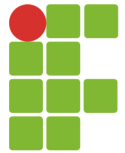
Laços Aninhados: Soma de Pares

```c
#include <stdio.h>

int main(void){
    int c;
    int i,j;
    printf("Digite o valor de c: ");
    scanf("%d",&c);
    for(i=1;i<=100;i++){
        for(j=1;j<=100;j++){
            if(i+j == c){
                printf("%d + %d = %d\n",i,j,c);
            }
        }
    }
    return 0;
}
```



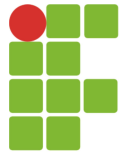
Laços Aninhados

- E se quiséssemos imprimir todos os pares **distintos** com tal propriedade?
- Basta fazer com que `j=i` na inicialização do laço interno.



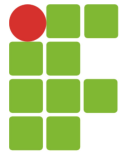
Laços Aninhados: Soma de Pares

```c
#include <stdio.h>

int main(void){
    int c;
    int i,j;
    printf("Digite o valor de c: ");
    scanf("%d",&c);
    for(i=1;i<=100;i++){
        for(j=i;j<=100;j++){
            if(i+j == c){
                printf("%d + %d = %d\n",i,j,c);
            }
        }
    }
    return 0;
}
```

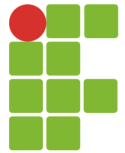


Laços Aninhados

- Veremos agora uma série de problemas em que podemos aplicar laços aninhados em suas soluções.

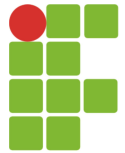


Sumário



Laços Aninhados: Dados

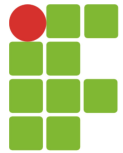
- Problema: imprimir todas as possíveis combinações do lançamento de 6 dados com 6 faces cada um.



Laços Aninhados: Dados

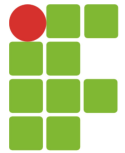
Estratégia

- 6 laços aninhados.
- Cada laço ficará responsável pelo valor de um determinado dado.

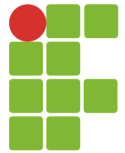


Laços Aninhados: Dados

```c
#include <stdio.h>

int main(void) {
    int dado_1, dado_2, dado_3, dado_4, dado_5, dado_6;
    for (dado_1 = 1; dado_1 <= 6; dado_1++) {
        for (dado_2 = 1; dado_2 <= 6; dado_2++) {
            for (dado_3 = 1; dado_3 <= 6; dado_3++) {
                for (dado_4 = 1; dado_4 <= 6; dado_4++) {
                    for (dado_5 = 1; dado_5 <= 6; dado_5++) {
                        for (dado_6 = 1; dado_6 <= 6; dado_6++) {
                            printf("Lançamento: %d %d %d %d %d %d\n", dado_1,
                                   dado_2, dado_3, dado_4, dado_5, dado_6);
                        }
                    }
                }
            }
        }
    }
    return 0;
}
```

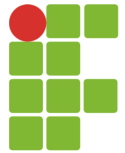


Sumário



Laços Aninhados: Números Primos

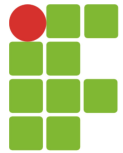
- Problema: dado um inteiro $n \geq 0$, imprimir os n primeiros números primos.



Laços Aninhados: Números Primos

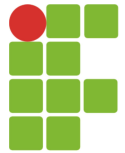
Estratégia

- Já sabemos detectar se um número é ou não é primo.
- A estratégia aqui é manter uma variável contadora `cnt` e atualizá-la toda vez que um primo for encontrado até que ela chegue ao valor n.
- Devemos checar se a variável `candidato` contém ou não um número primo. Ela começa de 2 e é incrementada antes de cada verificação de número primo.

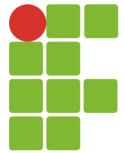


Laços Aninhados: Números Primos

```c
#include <math.h>
#include <stdio.h>

int main(void) {
    int i, cnt, n, candidato, primo;
    printf("Informe a quantidade de números primos a serem impressos: ");
    scanf("%d", &n);
    for (cnt = 0, candidato = 2; cnt < n; candidato++) {
        primo = 1;
        for (i = 2; i <= sqrt(candidato) && primo; i++) {
            if (candidato % i == 0)
                primo = 0;
        }
        if (primo) {
            printf("%d ", candidato);
            cnt++;
        }
    }
    printf("\n");
    return 0;
}
```

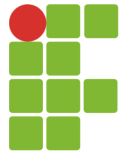


Sumário



Interrupção em laços aninhados

- É importante ressaltar que os comandos `break` e `continue` aplicam-se apenas ao laço em que foi aplicado.
- Deve-se ter cuidado de onde colocá-los para obtenção do comportamento desejado.



Laços Aninhados: Números Primos

```c
#include <math.h>
#include <stdio.h>
int main(void) {
    int i, cnt, n, candidato, primo;
    printf("Informe a quantidade de números primos a serem impressos: ");
    scanf("%d", &n);
    for (cnt = 0, candidato = 2; cnt < n; candidato++) {
        primo = 1;
        for (i = 2; i <= sqrt(candidato); i++) {
            if (candidato % i == 0) {
                primo = 0;
                break;
            }
        }
        if (primo) {
            printf("%d ", candidato);
            cnt++;
        }
    }
    printf("\n");
    return 0;
}
```