Algoritmos e Estruturas de Dados I

1º Período Engenharia da Computação

Prof. Edwaldo Soares Rodrigues
Email: edwaldo.rodrigues@uemg.br
Material adaptado do prof. André Backes

UNIVERSIDADE
DO ESTADO DE MINAS GERAIS | UEMG
UNIDADE DIVINÓPOLIS

Comandos de Repetição

- Uma estrutura de repetição permite que uma sequência de comandos seja executada repetidamente, enquanto determinadas condições são satisfeitas.

- Essas condições são representadas por expressões lógica (como, por exemplo, A>B; C==3; Letra == ‘a’)
  - Repetição com Teste no Início
  - Repetição com Teste no Final
  - Repetição Contada

Comandos de Repetição

- O real poder dos computadores está na sua habilidade para repetir uma operação ou uma serie de operações muitas vezes.

- Este repetição chamada **laços** (loop) é um dos conceitos básicos da programação estruturada

Repetição por Condição

- Um conjunto de comandos de um algoritmo pode ser repetido quando subordinado a uma condição:

enquanto *condição* **faça**
  comandos;
fim enquanto

- De acordo com a condição, os comandos serão repetidos zero (se falso) ou mais vezes (enquanto a condição for verdadeira).
  - Essa estrutura normalmente é denominada **laço** ou **loop**

Repetição por Condição

- Condição
  - qualquer expressão que resulte em um valor do tipo lógico e pode envolver operadores aritméticos, lógicos, relacionais e resultados de funções.
  - Ex:

    x > 15;

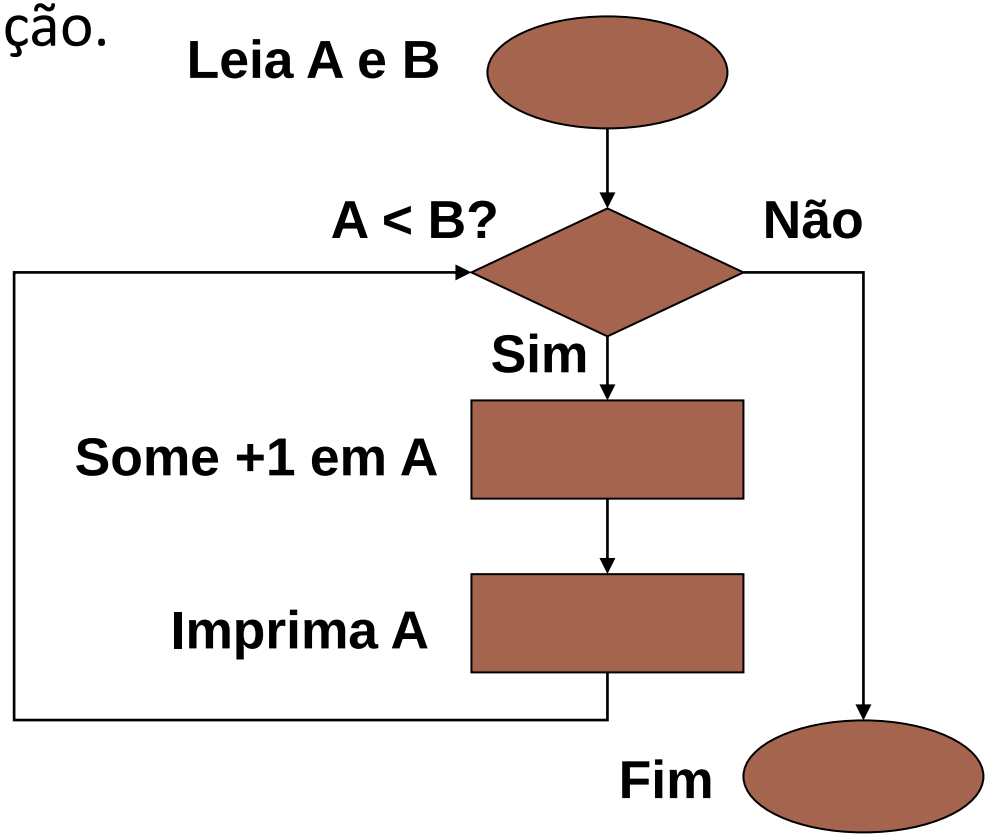
    (N < 10) && (N > 15);

Funcionamento

- A condição da cláusula ***enquanto*** é testada.
  - Se ela for verdadeira os comandos seguintes são executados em sequência como em qualquer algoritmo, até a cláusula ***fim enquanto***.
  - O fluxo nesse ponto é desviado de volta para a cláusula ***enquanto*** e o processo se repete.
  - Se a condição for falsa (ou quando finalmente for), o fluxo do algoritmo é desviado para o primeiro comando após a cláusula ***fim enquanto***.

Repetição por Condição

- Relembrando em fluxogramas
  - Um processo pode ser repetido até atender ou não uma condição.



Loop Infinito

- Um loop ou laço infinito ocorre quando cometemos algum erro
  - ao especificar a condição lógica que controla a repetição;
  - ou por esquecer de algum comando dentro da iteração;

Loop Infinito

Condição errônea

```
X recebe 4;
enquanto (X < 5) faça
    X recebe X – 1;
    Imprima X;
fim enquanto
```

Não muda valor

```
X recebe 4;
enquanto (X < 5) faça
    Imprima X;
fim enquanto
```

Comando while

- Equivale ao comando “enquanto” utilizado nos pseudo-códigos.
  - Repete a sequência de comandos enquanto a condição for verdadeira;
  - **Repetição com Teste no Início;**

- Esse comando possui a seguinte forma geral:

```
while (condição) {
    sequência de comandos;
}
```

Comando while - exemplo

- Faça um programa que mostra na tela os número de 1 a 100:

```c
int main(){
    // programa que mostra na tela números de 1 ate 100
    printf(" 1 2 3 4 .... ");
    return 0;
}
```

- A solução acima é inviável para valores grandes. Precisamos de algo mais eficiente e inteligente;

Comando while - exemplo

- Faça um programa que mostra na tela os número de 1 a 100:

```
int main(){
    // programa que mostra na tela números de 1 ate 100
    int numero;
    numero = 1;
    while(numero <= 100){
        printf("%d", numero);
        numero = numero + 1;
    }
    return 0;
}
```

Inicializa o contador

Incrementa o contador

- Observe que a variável **numero** é usada como um **contador**, ou seja, vai contar quantas vezes o loop será executado;

Comando while - exemplo

- Faça um programa para ler 5 números e mostrar o resultado da soma desses números:

```c
int main(){
    float val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%f", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%f", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%f", &val3);

    printf("\nDigite o 4o. numero: ");
    scanf("%f", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%f", &val5);

    soma = val1 + val2 + val3 + val4 + val5;
    printf("\nO resultado da soma eh: %f", soma);

    return 0;
}
```

Comando while - exemplo

- Faça um programa para ler 5 números e mostrar o resultado da soma desses números:

```
int main(){
    float val, soma;
    int contagem;
    // inicializando o valor de soma
    soma = 0;
    // inicializando o contador
    contagem = 1;
    while(contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%f", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }
    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Acumulador

Acumula a soma a cada passo do loop

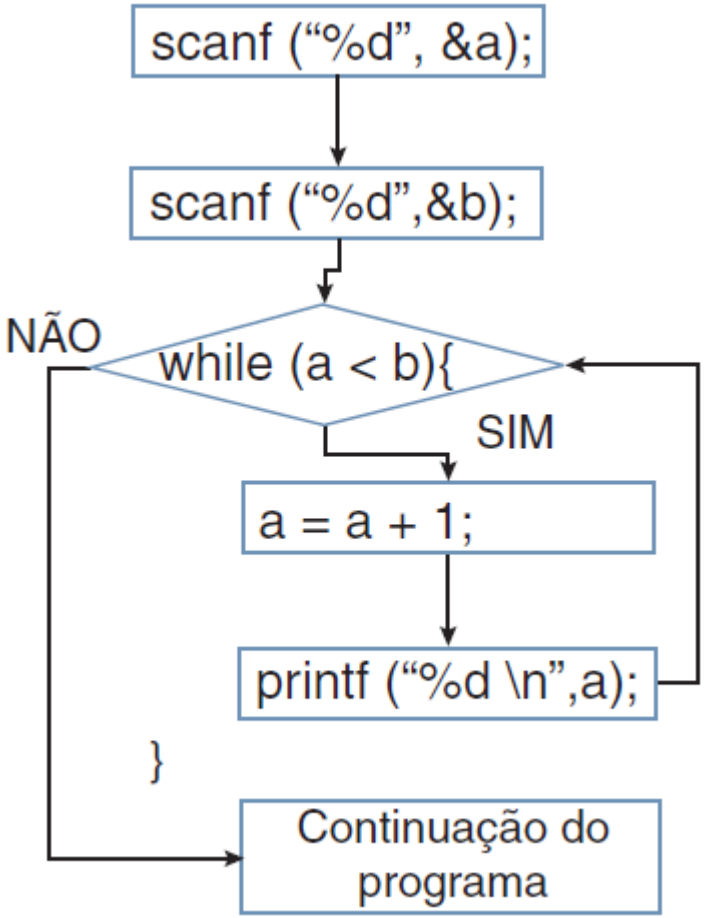
Controla o número de execuções

Comando while - exemplo

- Imprimindo os números entre A e B:

```
int main(){
    int a, b;
    printf("Digite o valor de a:");
    scanf("%d",&a);
    printf("Digite o valor de b:");
    scanf("%d",&b);

    while(a < b){
        a = a + 1;
        printf("%d \n", a);
    }

    return 0;
}
```

Comando while - exemplo



Comando do-while

- Comando **while**: é utilizado para repetir um conjunto de comandos zero ou mais vezes;
  - Repetição com Teste no Início;
- Comando **do-while**: é utilizado sempre que o bloco de comandos **deve ser executado ao menos uma vez.**
  - **Repetição com Teste no Final;**

Comando do-while

- Executa comandos;
- Avalia condição:
  - se verdadeiro, reexecuta bloco de comandos;
  - caso contrário, termina o laço;

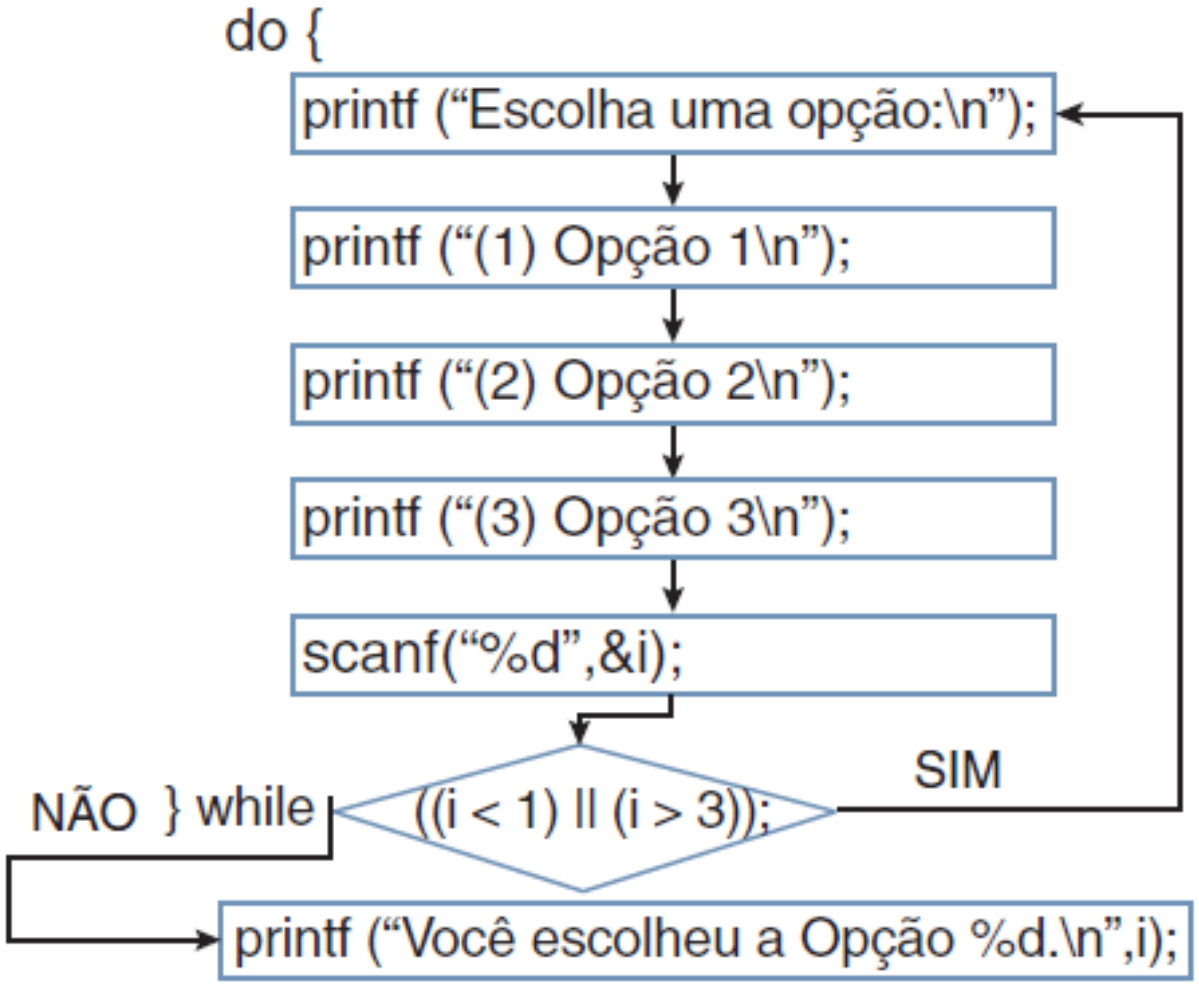
- Sua forma geral é (sempre termina com ponto e vírgula!);

```
do {
    sequência de comandos;
} while (condição);
```

Comando do-while

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    do{
        printf("Escolha uma opcao:\n");
        printf("(1) Opcao 1\n");
        printf("(2) Opcao 2\n");
        printf("(3) Opcao 3\n");
        scanf("%d",&i);

    }while((i < 1) || (i > 3));

    system("pause");
    return 0;
}
```

Comando do-while



Comando for

- O loop ou laço ***for*** é usado para repetir um comando, ou bloco de comandos, diversas vezes;
  - Maior controle sobre o loop;
- Sua forma geral é

```
for(inicialização; condição; incremento){
    sequência de comandos;
}
```

Comando for

1. **Inicialização**: iniciar variáveis (contador);

2. **Condição**: avalia a condição. Se verdadeiro, executa comandos do bloco, senão encerra laço;

3. **Incremento**: ao término do bloco de comandos, incrementa o valor do contador;

4. Repete o processo até que a **condição** seja falsa;

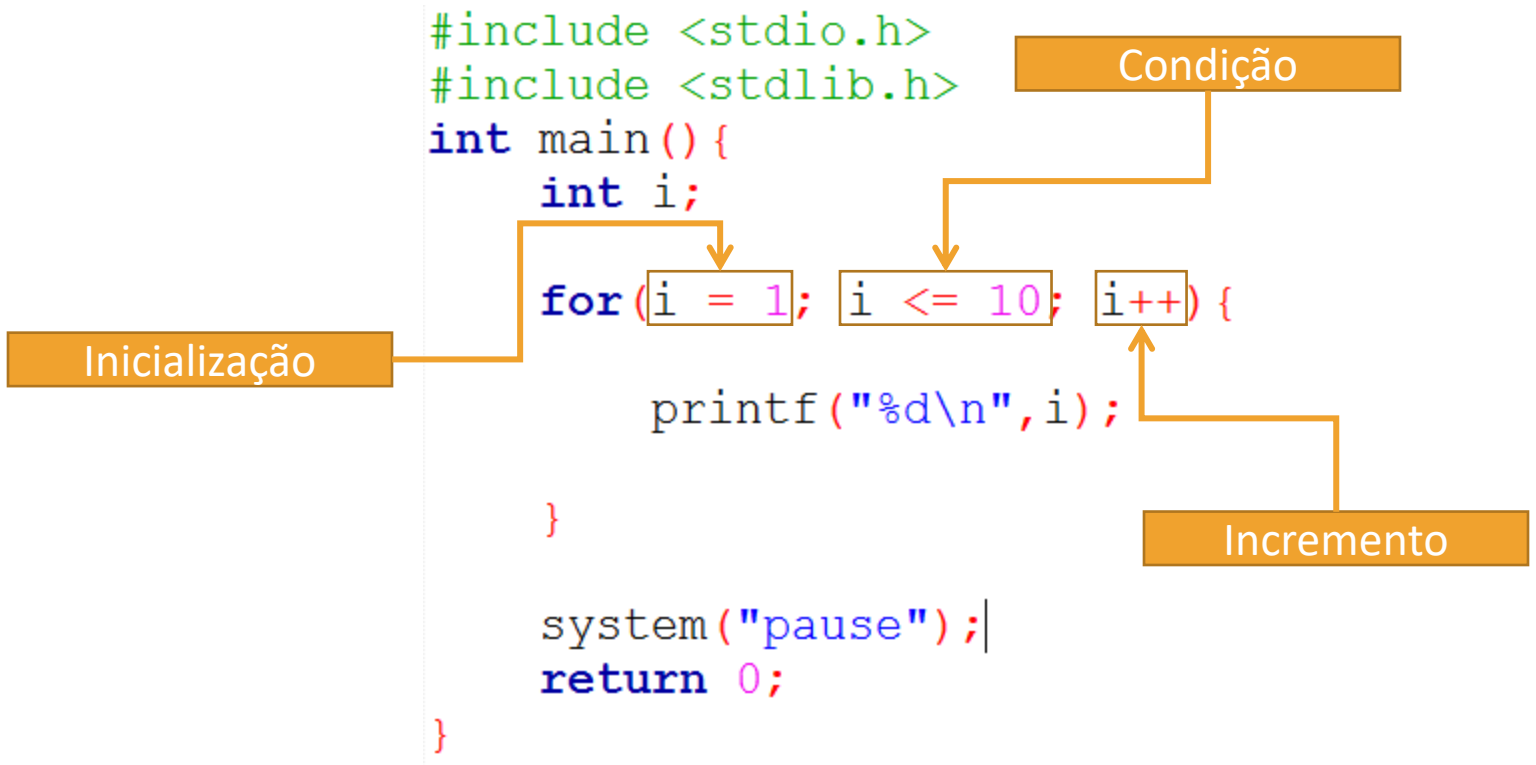
```
for(inicialização; condição; incremento){
    sequência de comandos;
}
```

Comando for

- Em geral, utilizamos o comando **for** quando precisamos ir de um valor inicial até um valor final;
- Para tanto, utilizamos uma variável para a realizar a contagem;
  - Exemplo: **int i**;
- Nas etapas do comando **for:**
  - Inicialização: atribuímos o valor inicial a variável;
  - Condição: especifica a condição para continuar no *loop:*
    - Exemplo: seu valor final;
  - Incremento: atualiza o valor da variável usada na contagem;

Comando for

- Exemplo: imprime os valores de 1 até 10



Comando for

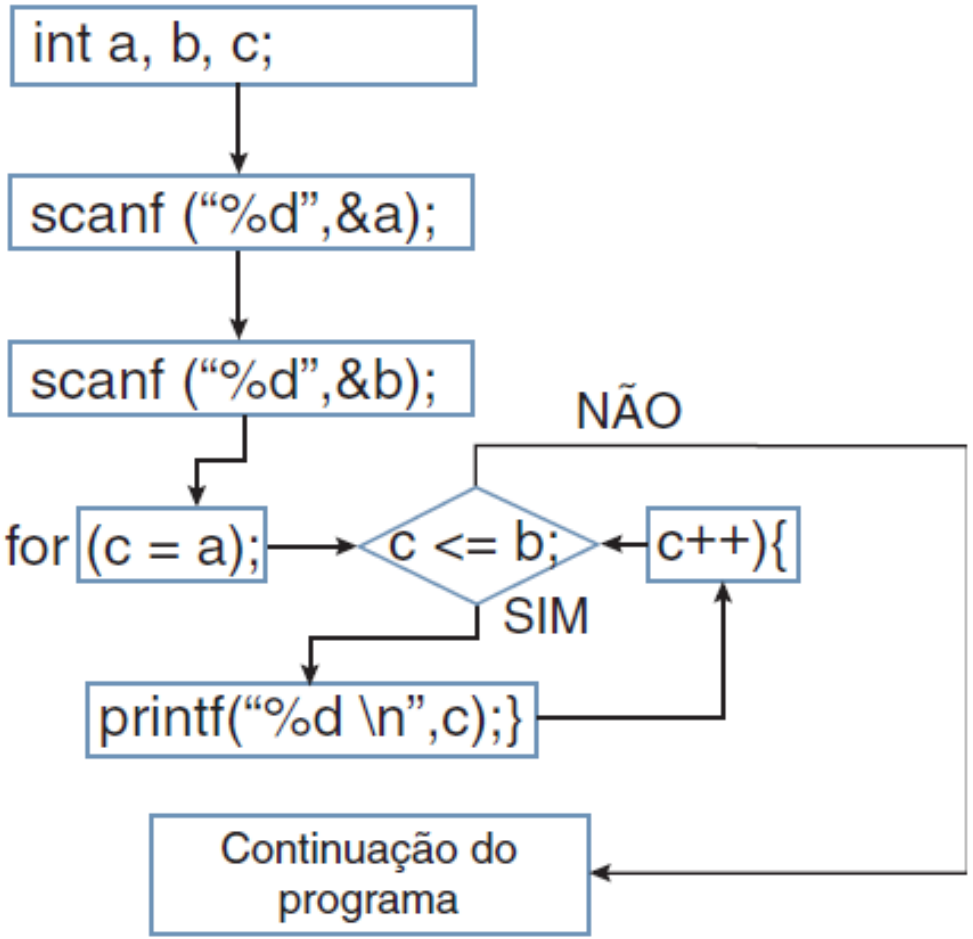
- Comando **while**: repete uma sequência de comandos enquanto uma condição for verdadeira;

- Comando **for**: repete uma sequência de comandos “N vezes”;

Exemplo for

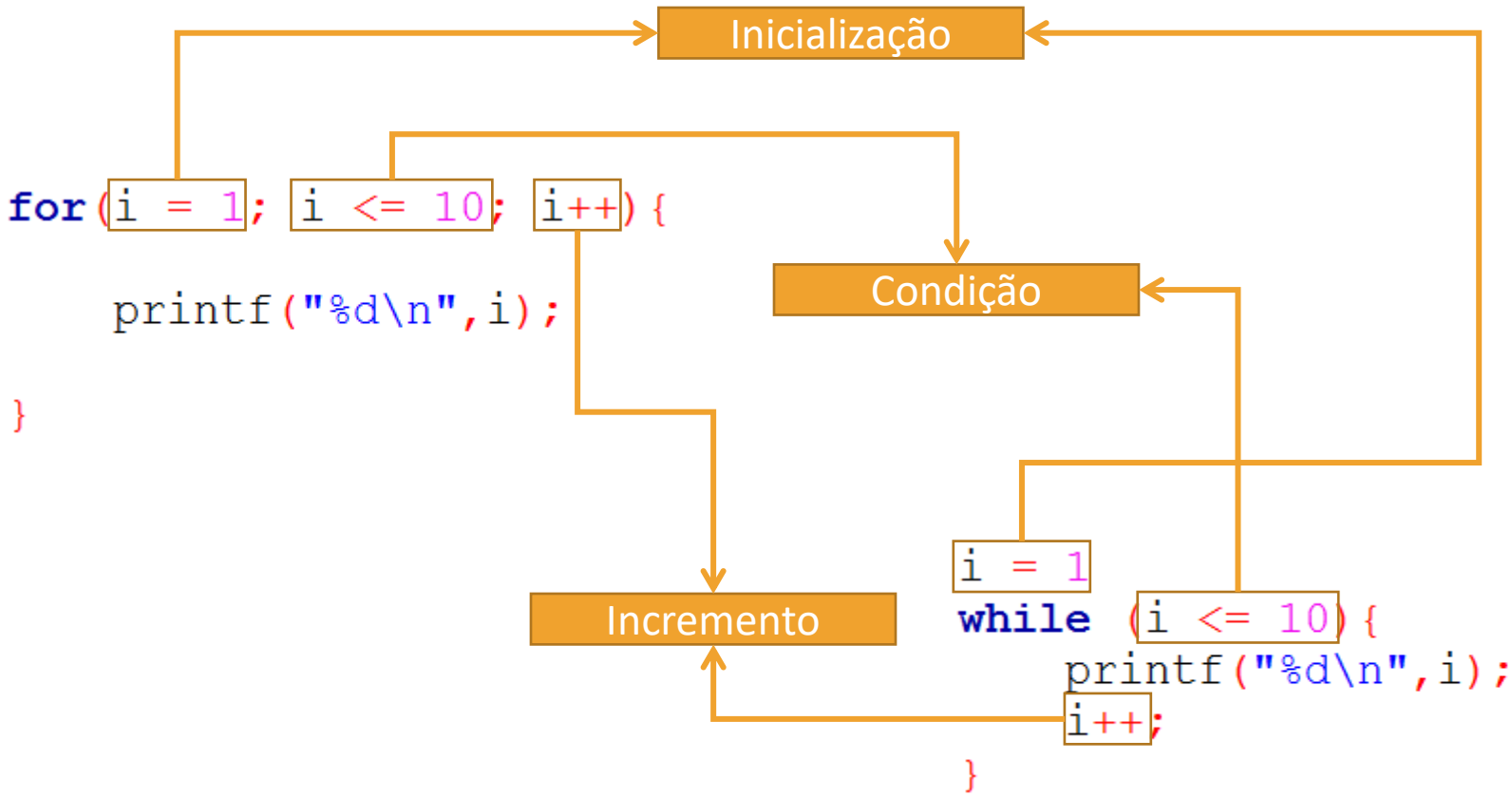
```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b,c;
    printf("Digite o valor de a: ");
    scanf("%d",&a);
    printf("Digite o valor de b: ");
    scanf("%d",&b);
    for(c = a; c <= b; c++){
        printf("%d \n",c);
    }

    return 0;
}
```

Exemplo for


int a, b, c;
scanf (“%d”,&a);
scanf (“%d”,&b);
for (c = a);
c <= b;
c++){
NÃO
SIM
printf(“%d \n”,c);}
Continuação do programa

For *vs* While

- Exemplo: mostra os valores de 1 até 10



Comando for

- Podemos omitir qualquer um de seus elementos:
  - Inicialização, condição ou incremento;
- Ex.: **for** sem inicialização:

```
#include <stdio.h>
#include <stdlib.h>
int main(){
  int a,b,c;
  printf("Digite o valor de a: ");
  scanf("%d",&a);
  printf("Digite o valor de b: ");
  scanf("%d",&b);
  for (; a <= b; a++){
    printf("%d \n",a);
  }
  system("pause");
  return 0;
}
```

Comando for

- Cuidado: for sem condição:
  - Omitir a condição cria um laço infinito;
  - Condição será sempre verdadeira;

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
  int a,b,c;
  printf("Digite o valor de a: ");
  scanf("%d",&a);
  printf("Digite o valor de b: ");
  scanf("%d",&b);
  //o comando for abaixo é um laço infinito
  for (c = a; ; c++){
    printf("%d \n",c);
  }
  system("pause");
  return 0;
}
```

Comando for

- Cuidado: for sem incremento:
  - Omitir o incremento cria um laço infinito;
  - Incremento pode ser feito nos comandos;

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
  int a,b,c;
  printf("Digite o valor de a: ");
  scanf("%d",&a);
  printf("Digite o valor de b: ");
  scanf("%d",&b);
  for (c = a; c <= b; ){
    printf("%d \n",c);
    c++;
  }
  system("pause");
  return 0;
}
```

Comando break

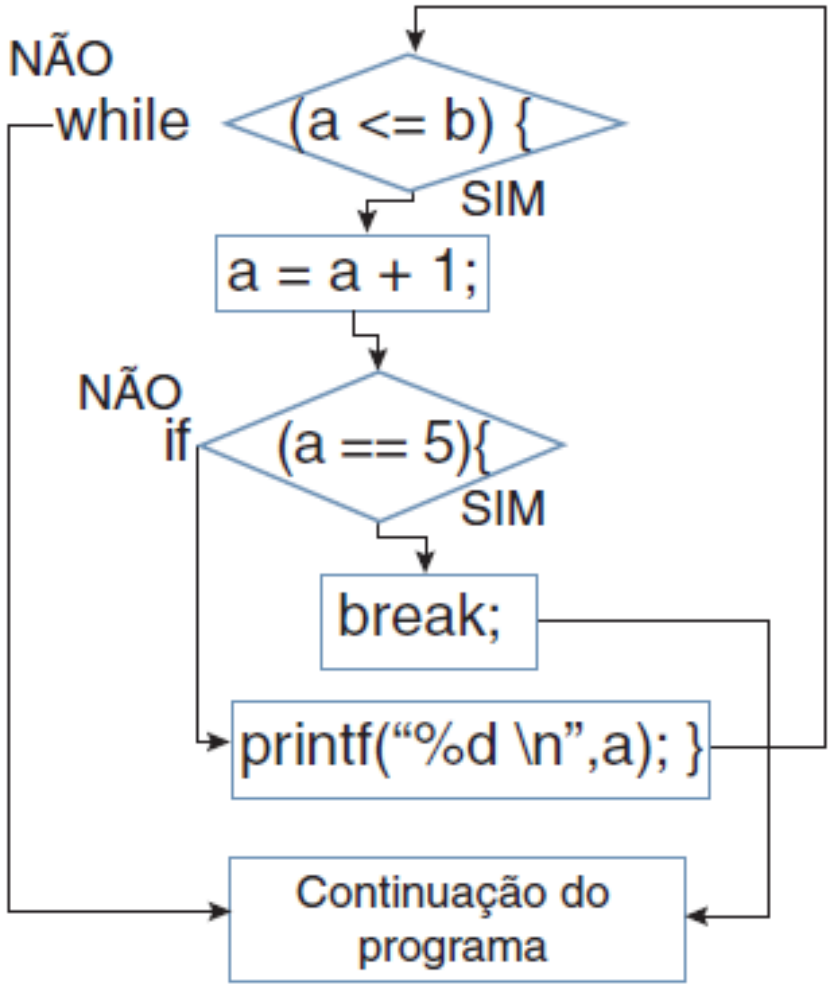
- Na verdade, o comando **break** serve para:
  - quebrar a execução de um comando (como no caso do **switch**);
  - interromper a execução de qualquer *loop* (**for**, **while** ou **do-while**);

- O comando **break** é utilizado para terminar de forma abrupta uma repetição. Por exemplo, se estivermos dentro de uma repetição e um determinado resultado ocorrer, o programa deverá sair da repetição e continuar na primeira linha seguinte a ela;

Comando break

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    int a,b;
    printf("Digite o valor de a: ");
    scanf("%d",&a);
    printf("Digite o valor de b: ");
    scanf("%d",&b);
    while (a <= b){
        a = a + 1;
        if(a == 5)
            break;
        printf("%d \n",a);
    }

    return 0;
}
```

Comando break



Comandos de repetição aninhados

- Diz-se que um comando de repetição é aninhado quando há por exemplo:
  - Um for dentro de um for;
  - Um for dentro de um while;
  - Um for dentro de um do-while;
  - Um while dentro de um while;
  - Um while dentro de um for;
  - Um while dentro de um do-while;
  - Um do-while dentro de um while;
  - Um do-while dentro de um for;
  - Um do-while dentro de um do-while;

Comandos de repetição aninhados

- De um modo geral, sempre que há um comando de repetição dentro de outro comando de repetição diz-se que há um comando aninhado;

- Há muitas situações onde precisaremos fazer uso de comandos aninhados;

Algoritmos e Estruturas de Dados I

- Bibliografia:
  - Básica:
    - CORMEN, Thomas; RIVEST, Ronald, STEIN, Clifford, LEISERSON, Charles. Algoritmos. Rio de Janeiro: Elsevier, 2002.
    - DEITEL, Paul; DEITEL, Harvey. C++ como programar. 5. ed. São Paulo: Pearson, 2006.
    - MELO, Ana Cristina Vieira de; SILVA, Flávio Soares Corrêa da. Princípios de linguagens de programação. São Paulo: Edgard Blücher, 2003.
  - Complementar:
    - ASCENCIO, A. F. G. & CAMPOS, E. A. V. Fundamentos da programação de computadores. 2. ed. São Paulo: Pearson Education, 2007.
    - MEDINA, Marcelo, FERTIG, Cristina. Algoritmos e programação: teoria e prática. Novatec. 2005.
    - MIZRAHI, V. V.. Treinamento em linguagem C: módulo 1. São Paulo: Makron Books, 2008.
    - PUGA, S. & RISSETTI, G. Lógica de programação e estruturas de dados com aplicações em java. São Paulo: Prentice Hall, 2004.
    - ZIVIANI, Nívio. Projeto de Algoritmos com Implementação em Pascal e C. Cengage Learning. 2010.

Algoritmos e Estruturas de Dados I