9002 — Aula 08
Algoritmos e Programação de Computadores

Instituto de Engenharia – UFMT

Segundo Semestre de 2014

13 de outubro de 2014

Roteiro

Revisão

- Nas aulas anteriores, vimos como escrever programas capazes de executar comandos de forma linear, e, se necessário, tomar decisões com relação à executar ou não um bloco de comandos.

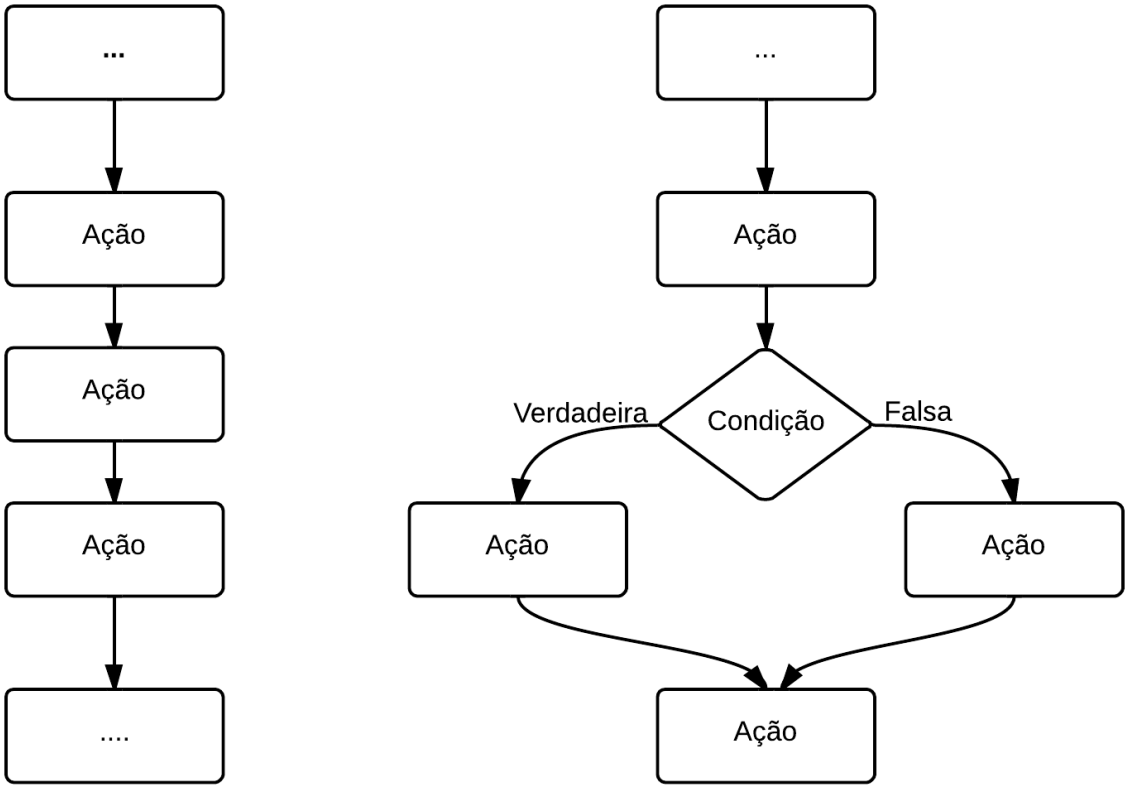

Fig 1 - Fluxogramas dos algoritmos das aulas anteriores.

Revisão

- Vimos, ainda, um procedimento de **simulação** (=*teste de mesa*), que pode e deve ser utilizado para testar o algoritmo.
- Esta técnica de simulação é composta apenas por dois passos:
  - Reservar espaço para cada variável
  - Executar em sequência cada um dos passos do algoritmo.

Exemplo — até 4

- Programa que imprime todos os números inteiros entre 1 e 4

```
...
printf("1");
printf("2");
printf("3");
printf("4");
...
```

Exemplo — até 4

- Programa que imprime todos os números inteiros entre 1 e 4

```
...
printf("1");
printf("2");
printf("3");
printf("4");
...
```

Exemplo — até 4

- Programa que imprime todos os números inteiros entre 1 e 4

```
...
printf("1");
printf("2");
printf("3");
printf("4");
...
```

Exemplo — até 4

- Programa que imprime todos os números inteiros entre 1 e 4

```
...
printf("1");
printf("2");
printf("3");
printf("4");
...
```

Exemplo — até 4

- Programa que imprime todos os números inteiros entre 1 e 4

```
...
printf("1");
printf("2");
printf("3");
printf("4");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até 100

- Programa que imprime todos os números inteiros entre 1 e 100

```
...
printf("1");
printf("2");
printf("3");
printf("4");
printf("5");
printf("6");
...
printf("99");
printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Exemplo — até *n* dado

- Programa que imprime todos os números inteiros entre 1 e *n* (dado). Sabendo que os valores de *n* variam entre 1 e 100.

```
...
int n;
scanf("%d", &n);
printf("1");
if (n>=2)
   printf("2");
if (n>=3)
   printf("3");
/* repete 96 vezes o bloco acima */
if (n>=100)
   printf("100");
...
```

Repetindo...

Se observarmos com atenção, perceberemos que, nestes casos, blocos de comandos são executados várias vezes (repetidamente) para obter o resultado.

Estruturas de repetição

- Permitem que uma sequência de comandos seja executada repetidas vezes. Cada uma destas execuções é chamada de *iteração*.
- A execução termina quando um critério de parada é atingido.
- Veremos o comando **while** que implementa uma construção do tipo **Enquanto-Faça**.

Construção *Enquanto-Faça*

- Nesta construção, o critério de parada é testado antes que a sequência de comandos da *ação* seja executada. Deste modo, a *ação* é executada **enquanto** a condição for verdadeira.

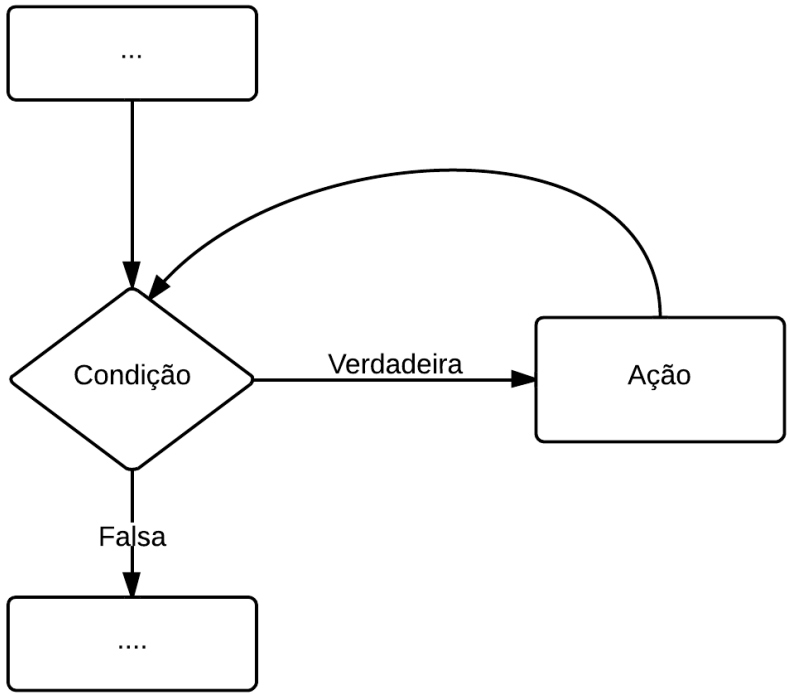

Fig. 2 - Fluxograma.

Construção *Enquanto-Faça*

Três questões são fundamentais:

- Qual *ação* deve ser repetida?
- Quantas vezes a repetição deve ser realizada?
- Qual condição pode ser utilizada para representar esta repetição?

Importante

Lembre-se de que um algoritmo deve sempre terminar. A condição projetada deve deixar de ser verdadeira em algum momento. Por isto, a *ação* deve alterar as variáveis envolvidas na condição de parada.

Construção *Enquanto-Faça*

Três questões são fundamentais:

- Qual *ação* deve ser repetida?
- Quantas vezes a repetição deve ser realizada?
- Qual condição pode ser utilizada para representar esta repetição?

Importante

Lembre-se de que um algoritmo deve sempre terminar. A condição projetada deve deixar de ser verdadeira em algum momento. Por isto, a *ação* deve alterar as variáveis envolvidas na condição de parada.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

Vamos refletir novamente sobre problema de imprimir os números inteiros entre 1 e 100.

- Qual *ação* deve ser executada?
  R: Impressão dos números 1, 2, 3, ... 100
  Ideia: Utilizar uma variável, por exemplo, i para representar os números. Iniciar i com valor 1. Incrementar i repetidamente.
- Quantas vezes a *ação* deve ser repetida?
  R. 100 vezes.
- Qual condição pode ser utilizada para representar esta repetição?
  R. Como a variável i indica quantas repetições foram executadas, podemos utilizá-la diretamente. Nossa condição pode ser, por exemplo, $i \leq 100$.

Construção *Enquanto-Faça*

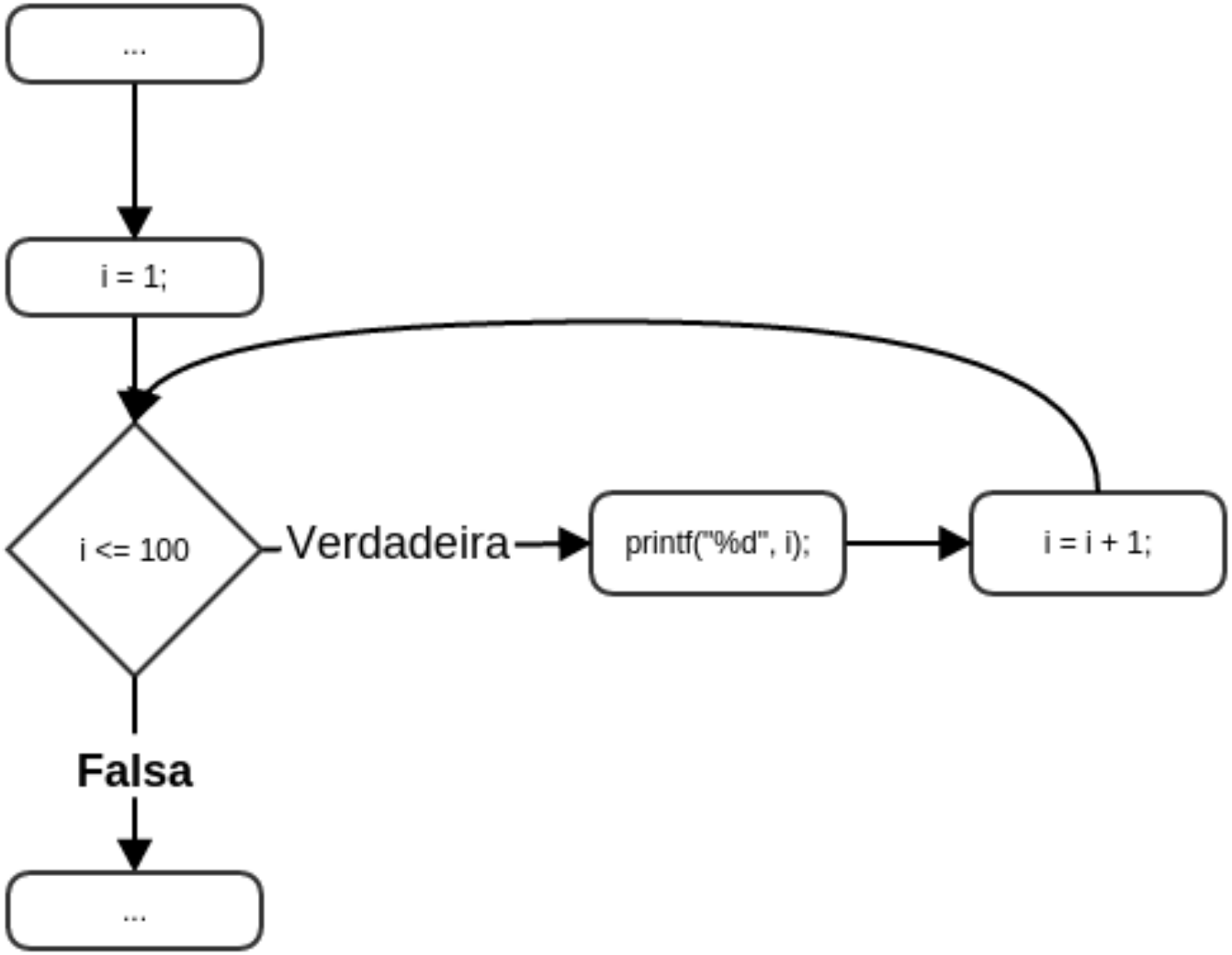

Fig. 3 - Fluxograma.

Comando while

O comando while é utilizado para executar um bloco de comandos enquando uma condição for satisfeita

Sintaxe do `while`

```
while (condicao) {
    ação;
}
```

Funcionamento:

- **Passo 1**: Testa condição:
  Se condição for verdadeira, executa o Passo 2.
  Se condição for falsa, executa o Passo 4 (pula o bloco de comandos).
- **Passo 2**: Executa o bloco de comandos.
- **Passo 3**: Volta para o Passo 1.
- **Passo 4**: Executa os comandos fora do bloco repetitivo.

Comando while

O comando while é utilizado para executar um bloco de comandos enquando uma condição for satisfeita

Sintaxe do `while`

```
while (condicao) {
    ação;
}
```

Funcionamento:

- **Passo 1**: Testa condição:
  Se condição for verdadeira, executa o Passo 2.
  Se condição for falsa, executa o Passo 4 (pula o bloco de comandos).
- **Passo 2**: Executa o bloco de comandos.
- **Passo 3**: Volta para o Passo 1.
- **Passo 4**: Executa os comandos fora do bloco repetitivo.

Imprimindo os 100 primeiros números inteiros.

```
...
i = 1;
while(i <= 100) {
   printf("%d", i);
   i = i + 1;
}
...
```

Imprimindo os 100 primeiros números inteiros.

```
...
i = 1;
while(i <= 100) {
   printf("%d", i);
   i = i + 1;
}
...
```

Imprimindo os 100 primeiros números inteiros.

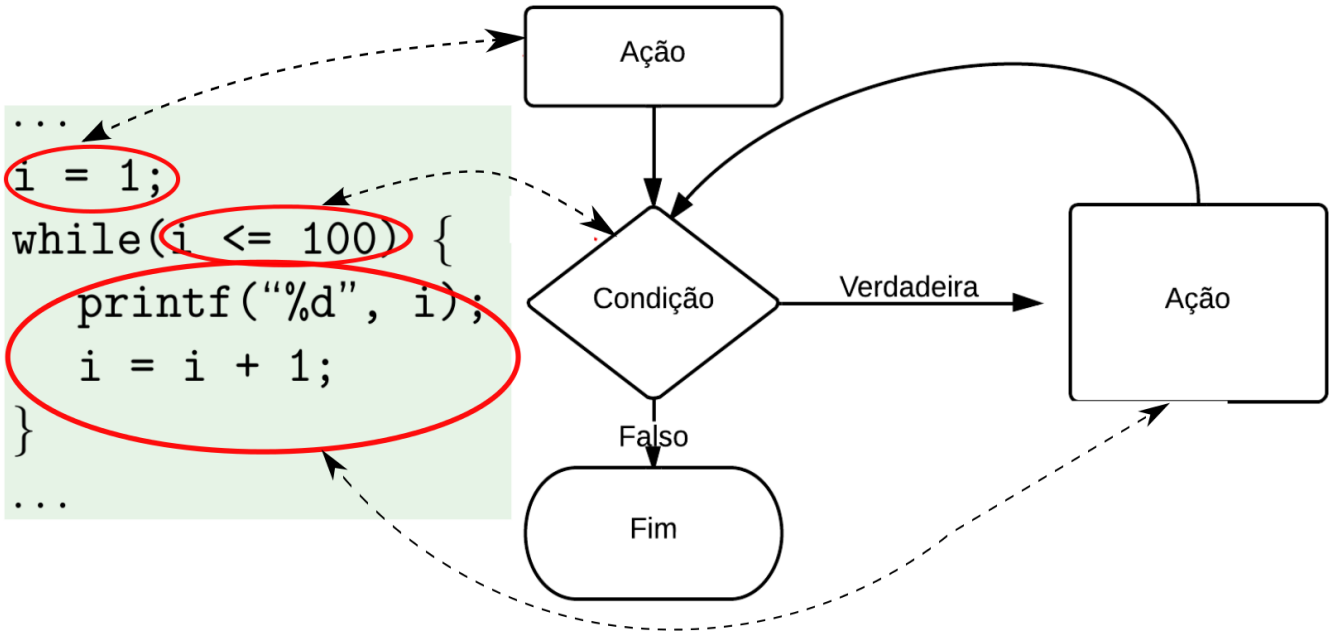

Fig. 3 - Relação entre o fluxograma e a construção.

Imprimindo os *n* primeiros números inteiros.

```
int i = 1, n;
scanf("%d", &n);
while (i <= n) {
    printf("%d ", i);
    i++;
}
```

Exercício

Exercício

Calcule a divisão inteira de dois numeros positivos usando apenas soma e subtração.

Solução — em C

```
...
scanf("%d %d", &dividendo, &divisor);

contador = 0;
while (dividendo >= divisor) {
  dividendo = dividendo - divisor;
  contador++;
}
...
```

Simulação Manual - Recapitulando

Simulando código

- Bem simples: Existem apenas 2 passos.
  - "Reservar" os espaços para os nossos objetos
  - "Executar" em sequência cada um dos passos do algoritmo.

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Reservando espaço para as variáveis

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Reservando espaço para as variáveis

Após executar a linha 1.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	21	7	?

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Reservando espaço para as variáveis

Após executar a linha 2.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	21	7	0

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.1.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	14	7	0

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.2.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	14	7	1

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.1.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	7	7	1

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.2

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	7	7	2

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.1.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	0	7	2

Simulando

```
...
scanf("%d %d", &dividendo, &divisor); /*1*/

contador = 0; /*2*/
while (dividendo >= divisor) { /*3*/
  dividendo = dividendo - divisor; /*3.1*/
  contador++; /*3.2*/
}
...
```

Após executar a linha 3.2.

Tipo	inteiro positivo	inteiro positivo	inteiro positivo
Nome	dividendo	divisor	contador
Valor	0	7	3

Simulação Manual - Exercício

Simulando código

Simule o algoritmo para *dividendo* igual à 35 e *divisor* igual à 6.

Solução — em C (completo)

```c
#include <stdio.h>
int main() {
  int dividendo, divisor, contador;

  // lê dividendo e divisor
  scanf("%d %d", &dividendo, &divisor);

  // realiza a divisão
  contador = 0;
  while (dividendo >= divisor) {
    dividendo = dividendo - divisor;
    contador++;
  }

  // mostra resultado da divisão
  printf("%d\n", contador);
  return 0;
}
```

Casos especiais do while

Casos especiais

1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a) a = a+1;
```

R: Ele nunca entra na repetição (*loop*).

2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a = a+1;
```

R: Ele entra na repetição e nunca sai (*loop* infinito).

O que acontece no programa de divisão quando o dividendo é menor que o divisor?

Casos especiais do while

Casos especiais

1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a) a = a+1;
```

R: Ele nunca entra na repetição (*loop*).

2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a = a+1;
```

R: Ele entra na repetição e nunca sai (*loop* infinito).

O que acontece no programa de divisão quando o dividendo é menor que o divisor?

Casos especiais do while

Casos especiais

1. O que acontece se a condição for falsa na primeira vez?

```
while (a!=a) a = a+1;
```

R: Ele nunca entra na repetição (*loop*).

2. O que acontece se a condição for sempre verdadeira?

```
while (a==a) a = a+1;
```

R: Ele entra na repetição e nunca sai (*loop* infinito).

O que acontece no programa de divisão quando o dividendo é menor que o divisor?

Exercício

1. Faça um programa que lê um número n e imprime o resultado da soma
$$\sum_{i=1}^{n} i$$
2. Faça um programa que lê um número n e imprime o fatorial de n (segundo exercício da lista).
3. Faça um programa que lê dois números inteiros n e a e imprime o resultado de a^n.

Nas próximas aulas...

Nas próximas aulas veremos:

- Outras construções iteartivas e seus respectivos comandos em C:
  1. Para-Faça - for
  2. Repita-Até - do...while
- Outros problemas cujas soluções envolvem o uso de comandos de repetição.