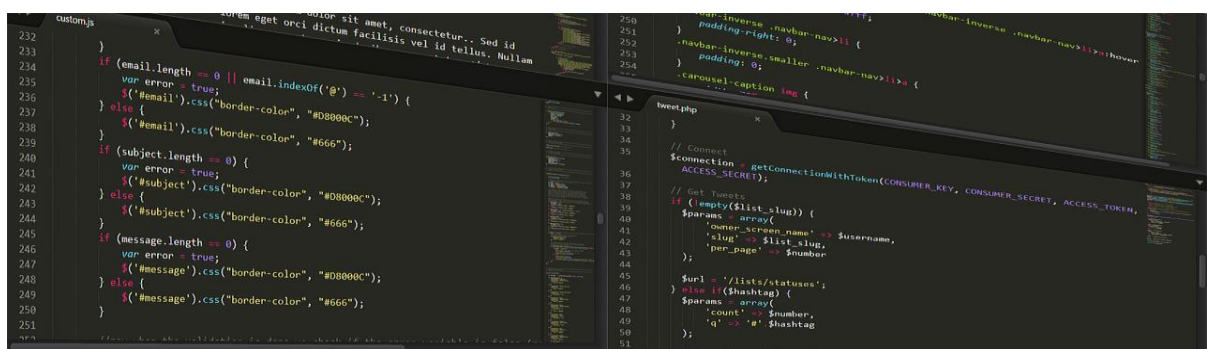




Lógica de Programação com Flowgorithm

Desenvolvedor de Salesforce



Sumário

Apresentação	4
1. Introdução à Lógica de Programação	5
1.1. O que é Lógica de Programação?	5
1.2. A Importância da Lógica de Programação	5
1.3. Desenvolvimento de Programas Passo a Passo	5
1.4. Aplicando Conceitos de Lógica de Programação	6
1.5. Linguagens de Programação	7
1.6. O que é Flowgorithm?	7
1.7. Português Estruturado	8
2. Variáveis, Constantes e Tipos de Dados	9
1.8. Variável	9
1.9. Constante	10
1.10. Tipos de Dados	11
1.11. Declaração e Inicialização de Variáveis	11
3. Entrada e Saída de Dados	14
1.12. Entrada de Dados	14
1.13. Saída de Dados	15
4. Operadores Matemáticos, Relacionais e Lógicos	18
1.14. Operadores Matemáticos	18
1.15. Operadores Relacionais	20
1.16. Operadores Lógicos	22
1.17. Exemplo Prático	24
5. Estruturas de Condição	25
1.18. Estrutura Condicional "Se"	25
1.19. Estrutura Condicional "Senão"	26
1.20. Estrutura Condicional "Senão Se"	27
1.21. Exemplo Prático	30
6. Estruturas de Repetição	31
6.1. Estrutura de Repetição "Para"	31
6.2. Estrutura de Repetição "Enquanto"	32

6.3.	Estrutura de Repetição "Faça Enquanto"	33
6.4.	Exemplo Prático	34
6.1.	Vetores	39
6.2.	Matrizes	40
7.	Funções e Procedimentos	42
7.1.	Funções	42
7.2.	Procedimentos.....	45
7.3.	Diferenças entre Funções e Procedimentos	47
	Exercícios Práticos	48

Apresentação

Sejam bem-vindos a aula de nivelamento de conceitos de Lógica de Programação com Flowgorithm, preparatório para o *Salesforce Platform Developer Credential*. Neste curso, iremos explorar os principais conceitos da Lógica de Programação utilizando como base a programação em português estruturado e com a criação de fluxogramas, com o software Flowgorithm.

Os principais conceitos a serem abordados são:

- Introdução a lógica de programação;
- Variáveis, constante e tipos de dados;
- Entradas e saídas de dados;
- Operadores matemáticos, relacionais e lógicos;
- Estruturas condicional e de seleção;
- Estruturas de Repetição;
- Vetores e Matrizes;
- Funções e Procedimentos.

1. Introdução à Lógica de Programação

1.1. O que é Lógica de Programação?

Lógica de programação é a técnica de planejar a sequência de instruções que um computador deve seguir para resolver um problema ou executar uma tarefa específica. É a base para a criação de algoritmos, que são conjuntos de passos ordenados que descrevem como realizar uma tarefa.

Um algoritmo é uma sequência de instruções que utilizamos para solucionar um ou vários problemas, ou até mesmo realizar tarefas do dia a dia.

Um algoritmo não é necessariamente um programa computacional, pode ser passos que iremos tomar para realizar determinada tarefa.

O algoritmo deve sempre chegar ao resultado final esperado, caso não chegue, o mesmo não pode ser considerado finalizado.

1.2. A Importância da Lógica de Programação

Entender a lógica de programação é essencial para desenvolver programas eficientes e eficazes. A lógica bem estruturada permite resolver problemas de forma clara e ordenada, tornando mais fácil a leitura, manutenção e atualização do código.

1.3. Desenvolvimento de Programas Passo a Passo

Desenvolver um programa envolve a criação de um algoritmo que define passo a passo as ações que o computador deve realizar. Esses passos são escritos em uma linguagem de programação, mas antes de chegarmos a isso, é crucial pensar na lógica por trás das ações.

Vamos considerar exemplos do mundo real para entender melhor o conceito de passos e instruções estruturadas.

Exemplo do Mundo Real: Fazendo um Café.

1. Preparar os Ingredientes:

- Pegue o café em pó.
- Pegue a água.
- Pegue o filtro de café.
- Pegue a cafeteira.

2. Ferver a Água:
 - Coloque a água na chaleira.
 - Ligue a chaleira e espere a água ferver.
3. Preparar o Filtro:
 - Coloque o filtro na cafeteira.
 - Adicione o café em pó no filtro.
4. Adicionar a Água:
 - Despeje a água fervida sobre o café no filtro.
5. Servir o Café:
 - Espere o café passar para a cafeteira.
 - Sirva o café na xícara.

Exemplo do Mundo Real: Trocando uma Lâmpada

1. Preparar o Ambiente:
 - Desligue o interruptor de luz.
 - Pegue uma nova lâmpada.
2. Remover a Lâmpada Queimada:
 - Suba em um banquinho, se necessário.
 - Gire a lâmpada queimada no sentido anti-horário até soltá-la do soquete.
3. Instalar a Nova Lâmpada:
 - Coloque a nova lâmpada no soquete.
 - Gire a nova lâmpada no sentido horário até fixá-la firmemente.
4. Testar a Nova Lâmpada:
 - Desça do banquinho, se necessário.
 - Ligue o interruptor de luz para verificar se a nova lâmpada está funcionando.

1.4. Aplicando Conceitos de Lógica de Programação

A lógica de programação segue princípios semelhantes aos exemplos do mundo real mencionados acima. Cada tarefa é decomposta em passos específicos que precisam ser seguidos em uma ordem lógica.

1.4.1. Sequência:

Cada instrução deve ser executada em uma ordem específica para alcançar o resultado desejado.

1.4.2. Decisão (Condicionais):

Em muitos casos, decisões precisam ser tomadas com base em condições.

Exemplo: "Se a água estiver fervendo, então despeje-a no filtro de café."

1.4.3. Repetição (Loops):

Algumas tarefas precisam ser repetidas várias vezes. **Exemplo:** "Despeje a água sobre o café até que toda a água tenha passado pelo filtro."

1.5. Linguagens de Programação

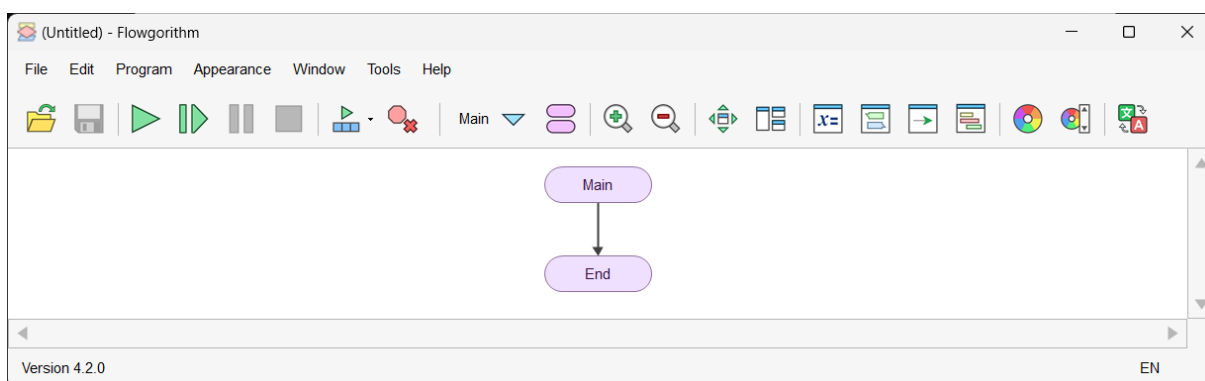
Existem muitas linguagens de programação, como Python, Java, C++, entre outras. Cada uma tem suas características e finalidades. Neste material, focaremos em conceitos gerais que são aplicáveis a diversas linguagens.

1.6. O que é Flowgorithm?

Flowgorithm é uma ferramenta visual que permite criar algoritmos usando diagramas de fluxo (fluxogramas). É uma ótima ferramenta para iniciantes, pois ajuda a entender a lógica de programação de forma visual e intuitiva.

Sua documentação pode ser acessada através do site, <http://www.flowgorithm.org/index.html>, onde também é possível realizar o download para desenvolvimento das atividades deste material.

Figura 1 – Tela inicial do Flowgorithm



1.7. Português Estruturado

Portugol, também conhecido como **Português estruturado**, é uma família de linguagens de programação que possui como base a língua portuguesa. Algumas de suas variações podem ser consideradas pseudocódigo, e outras são linguagens completas, livres de contexto, com gramáticas definidas e implementações em editores ou compiladores. São usadas tanto para o estudo de algoritmos e estruturas de dados quanto para a criação de compiladores, interpretadores e ferramentas de diagramação, como geradores de fluxogramas.

Exemplo:

```
algoritmo "ola-mundo"  
inicio  
    escreva ("Olá mundo")  
fimalgoritmo
```

Este capítulo forneceu uma introdução básica aos conceitos de lógica de programação, enfatizando a importância de pensar em termos de passos ordenados e estruturados.

No próximo capítulo, abordaremos variáveis, constantes e tipos de dados. Vamos explorar como armazenar e manipular dados em um programa. Fique atento para exemplos práticos e exercícios que ajudarão a fixar os conceitos aprendidos.

2. Variáveis, Constantes e Tipos de Dados

Neste capítulo, vamos aprender sobre variáveis, constantes e os diferentes tipos de dados que podemos usar em programação. Também veremos como declarar e usar esses elementos em nossos algoritmos.

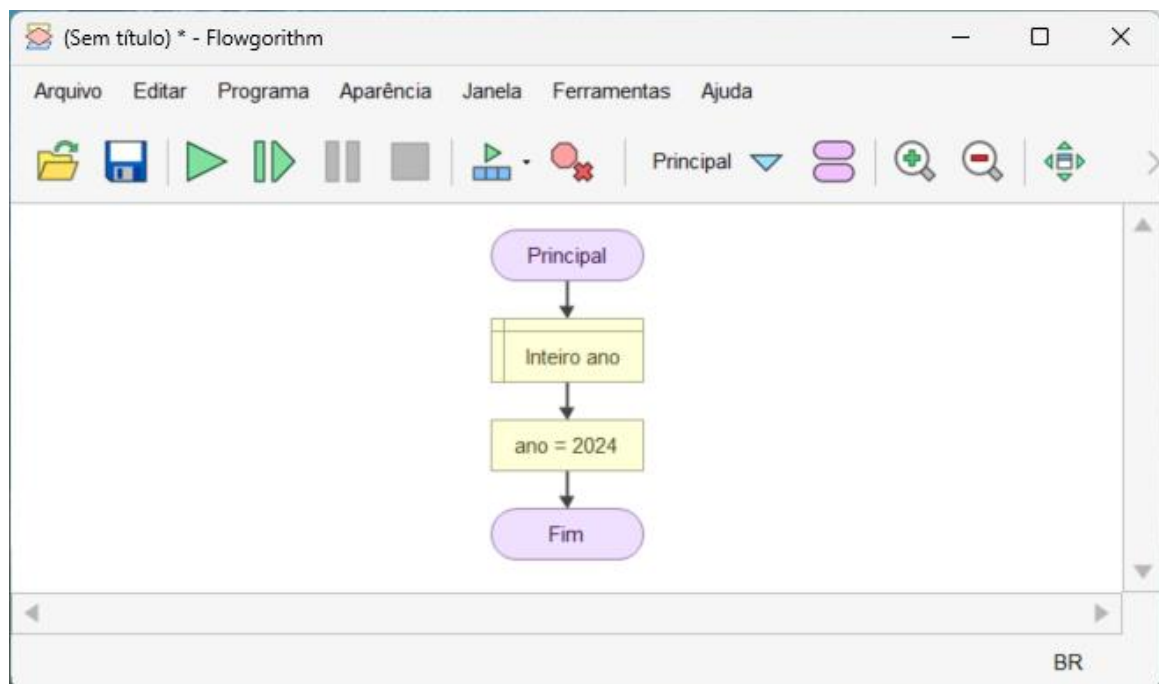
1.8. Variável

Variável é um recurso utilizado nos programas computacionais para armazenar e recuperar dados, ou seja, é simplesmente um espaço na memória que reservamos atribuindo um nome, para que possamos organizar os dados à serem manipulados pelo programa. Por exemplo, podemos criar uma variável chamada “idade” para armazenar a idade de uma pessoa. Seria como você ter uma gaveta de escritório com repartições, onde a gaveta em si, seria a memória e cada repartição uma variável para armazenar algum objeto.

Exemplo em Português Estruturado:

```
inteiro idade;  
real salario;  
caractere nome;
```

Exemplo no Flowgorithm:



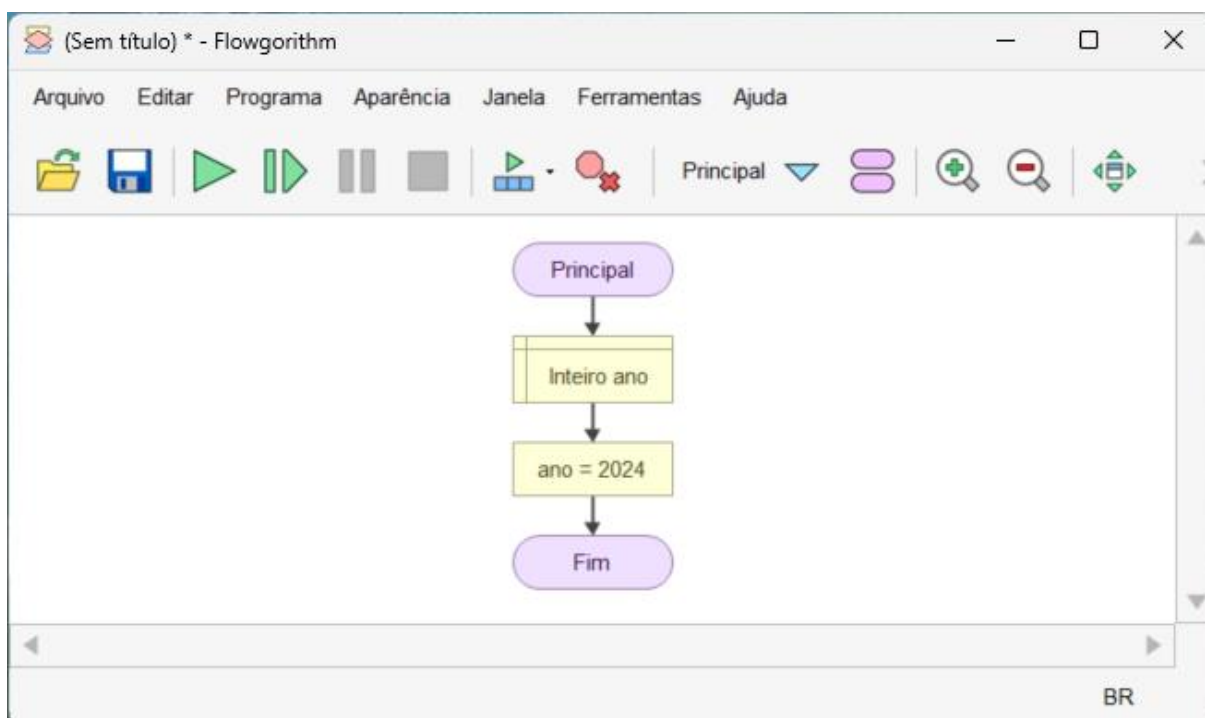
1.9. Constante

Uma **Constante** é responsável por armazenar um valor fixo em um espaço da memória, sendo que este valor não se altera durante a execução do programa. Um exemplo clássico para uso de constante é em relação ao valor do PI. O valor fixo aproximado de **PI** é **3.14159265359**. Sendo assim, em vez de sempre utilizarmos nas fórmulas o valor de **PI**, podemos definir uma constante que represente este valor durante a codificação.

Exemplo em Português Estruturado:

```
constante real PI = 3.14;
```

Exemplo no Flowgorithm:



No caso do **Flowgorithm**, ele possui internamente a constante **pi**, mas não possuem uma representação própria para constantes.

Desta forma, podemos dizer que: “**Variáveis e constantes são como caixas onde você guarda valores. Variáveis são caixas que você pode abrir e trocar o**

que tem dentro, enquanto constantes são caixas lacradas que não podem ser alteradas depois de fechadas."

1.10. Tipos de Dados

Os **Tipos de Dados** são a base para armazenar e manipular informações em qualquer linguagem de programação. Eles definem o tipo de valor que uma variável ou constante pode armazenar, como números inteiros, caracteres ou objetos.

Tipos de dados especificam o tipo de informação que uma variável pode armazenar. Os tipos de dados mais comuns são:

Inteiro: Números inteiros (ex.: -1, 0, 1, 2).

Real: Números reais ou de ponto flutuante (ex.: 3.14, -2.71).

Caractere: Um único caractere (ex.: 'a', 'b').

String: Uma sequência de caracteres (ex.: "Olá, Mundo!").

Booleano: Verdadeiro ou falso (ex.: verdadeiro, falso).

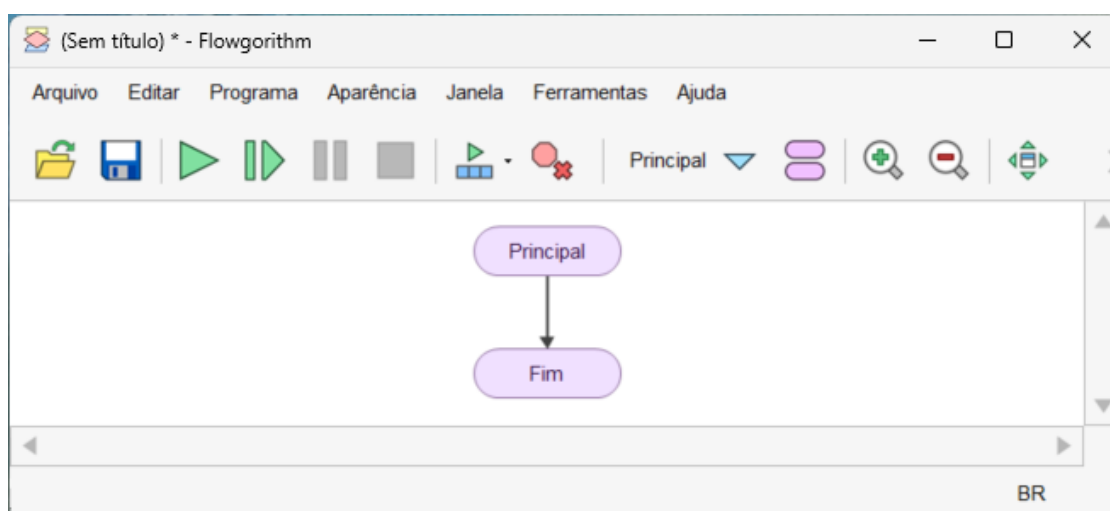
1.11. Declaração e Inicialização de Variáveis

Para usar uma variável, precisamos declará-la e, opcionalmente, inicializá-la com um valor.

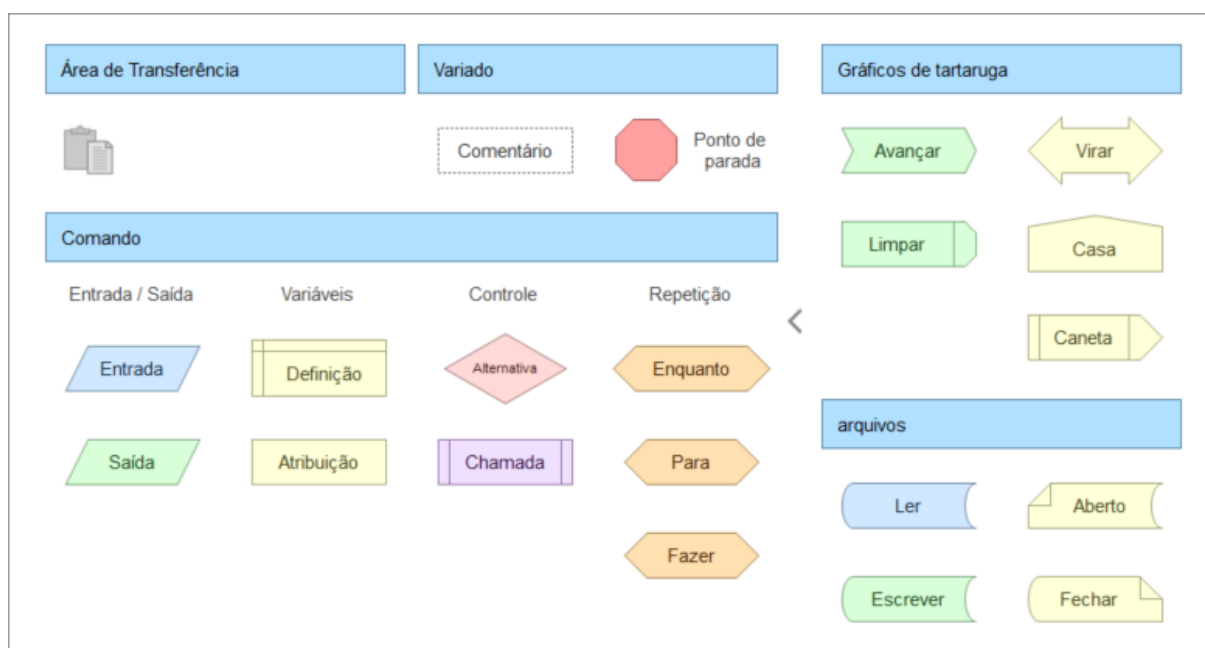
Exemplo em Português Estruturado:

```
inteiro idade = 25;
real salario = 3500.50;
caractere inicial = 'J';
string nome = "João";
booleano ativo = verdadeiro;
```

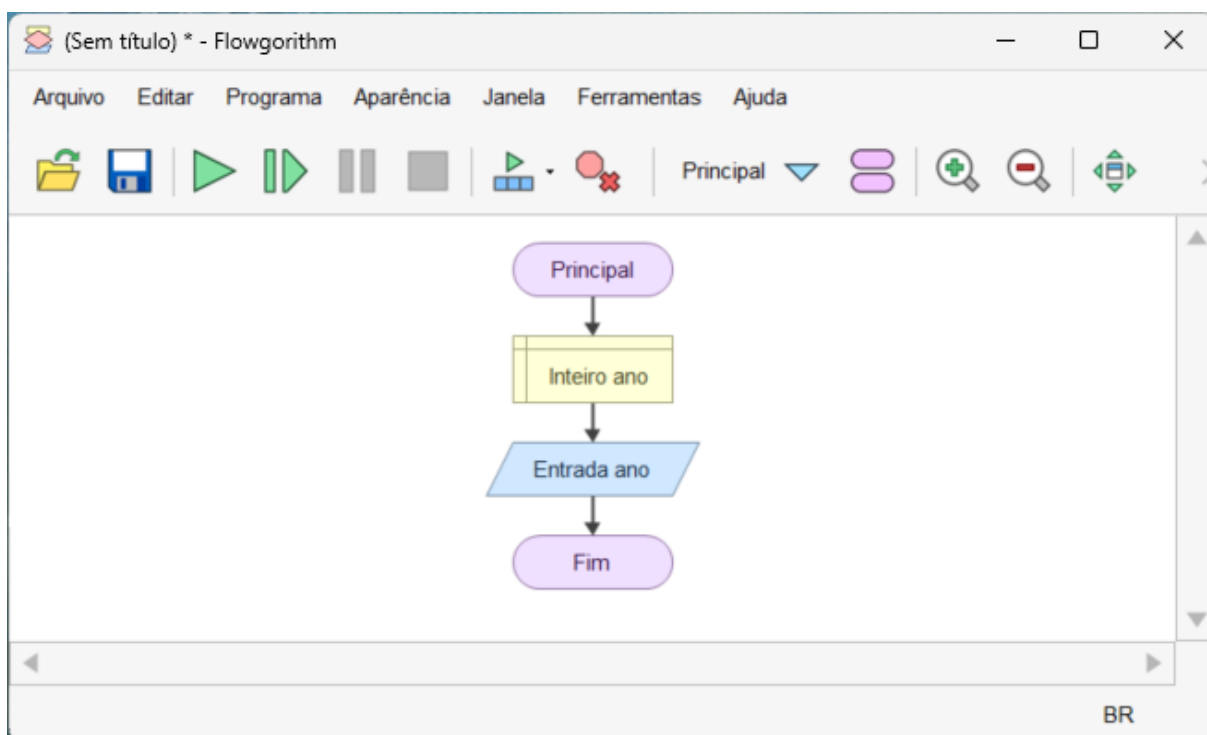
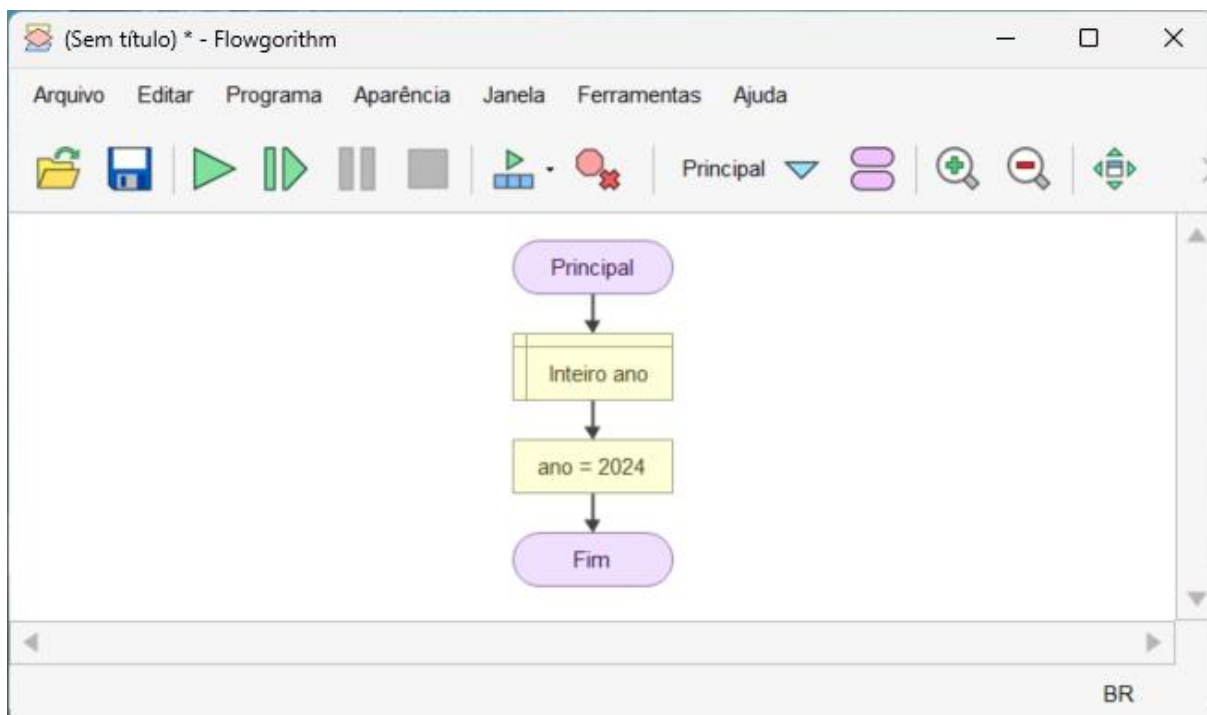
Exemplo no Flowgorithm:



Clique na seta, para abrir a janela de programação da ferramenta.



Nesta janela, é possível selecionar o comando que se deseja utilizar. Para variáveis, usamos a **Definição** para criar a variável e em seguida podemos usar a **Atribuição** para definir um valor inicial ou uma **Entrada** para solicitar ao usuário a digitação do valor.



No próximo capítulo, aprenderemos sobre entrada e saída de dados, explorando como interagir com o usuário para receber informações e exibir resultados. Prepare-se para mais exemplos práticos e exercícios desafiadores!

3. Entrada e Saída de Dados

Em programação, a entrada de dados é o processo de receber informações do usuário, enquanto a saída de dados é o processo de exibir informações para o usuário. Imagine um caixa eletrônico: você insere seu cartão (entrada) e o caixa mostra seu saldo (saída). Em um programa, a entrada e saída de dados são essenciais para a interação com o usuário.

1.12. Entrada de Dados

A entrada de dados é o processo de obter informações do usuário para que o programa possa processá-las. Em muitos algoritmos e programas, é essencial solicitar e receber dados do usuário para que as operações sejam realizadas com base nessas entradas.

Funções de Entrada:

leia(): Utilizada para ler um valor digitado pelo usuário.

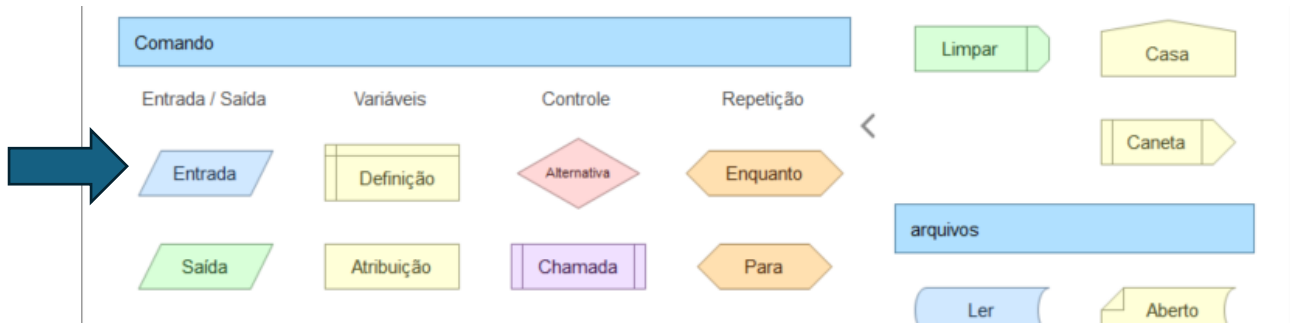
```
inteiro idade;
real salario;
string nome;

início
    escreva("Digite seu nome: ");
    leia(nome);

    escreva("Digite sua idade: ");
    leia(idade);

    escreva("Digite seu salário: ");
    leia(salario);

    escreva("Nome: ", nome, " Idade: ", idade, " Salário: ", salario);
fim
```



1.13. Saída de Dados

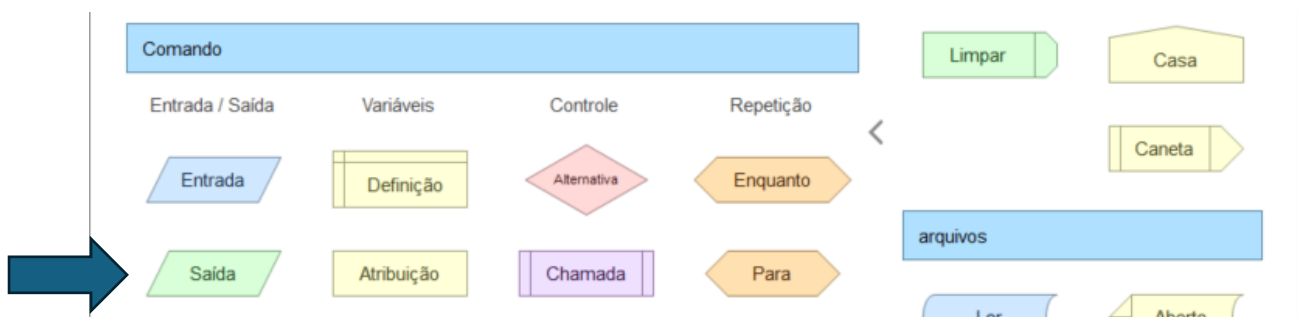
A saída de dados é o processo de mostrar informações ao usuário. Pode ser usada para exibir resultados de cálculos, mensagens informativas ou qualquer outro tipo de dado.

Funções de Saída:

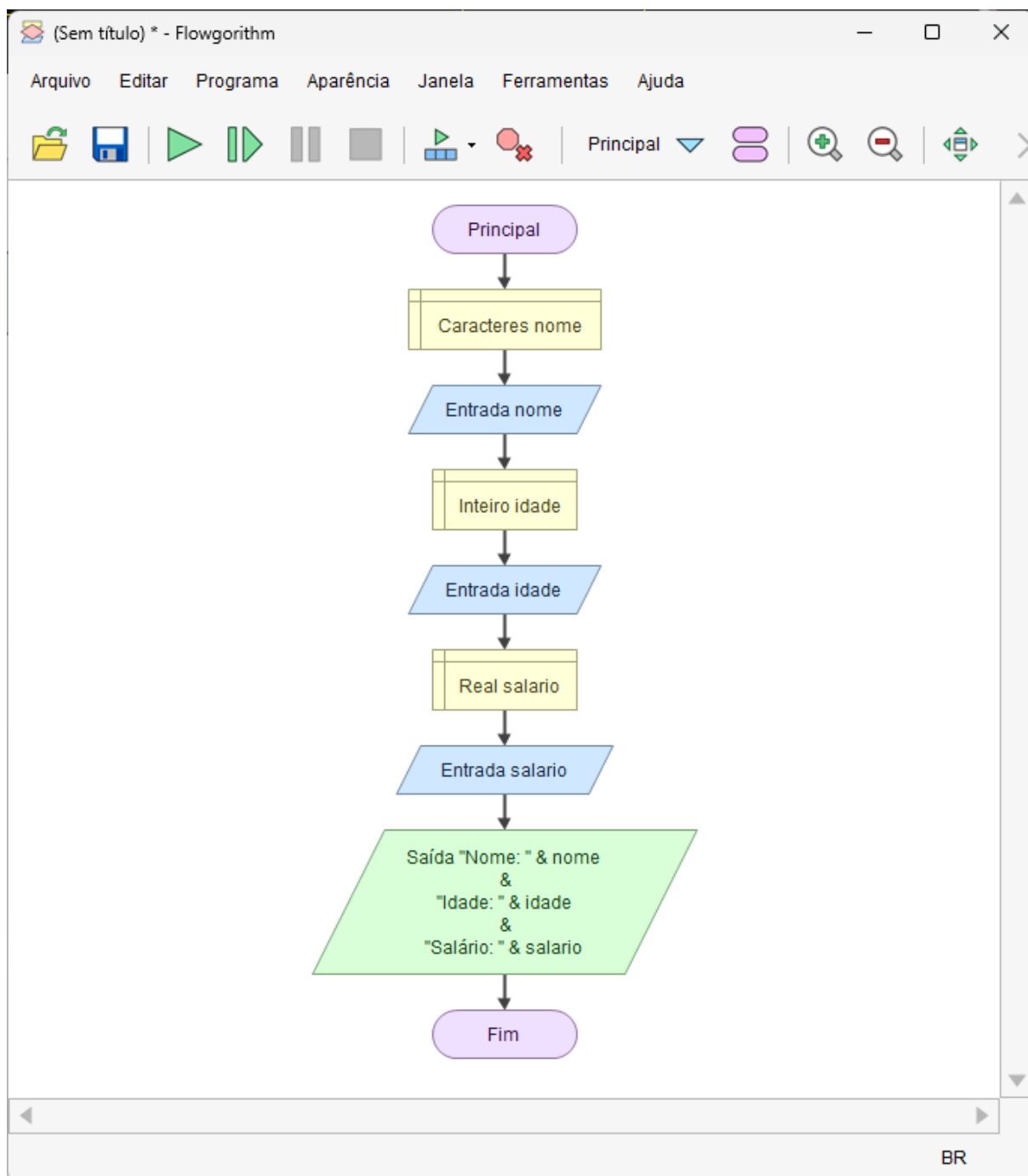
escreva(): Utilizada para exibir uma mensagem ou valor na tela.

```
inteiro idade = 25;
real salario = 3500.50;
string nome = "João";

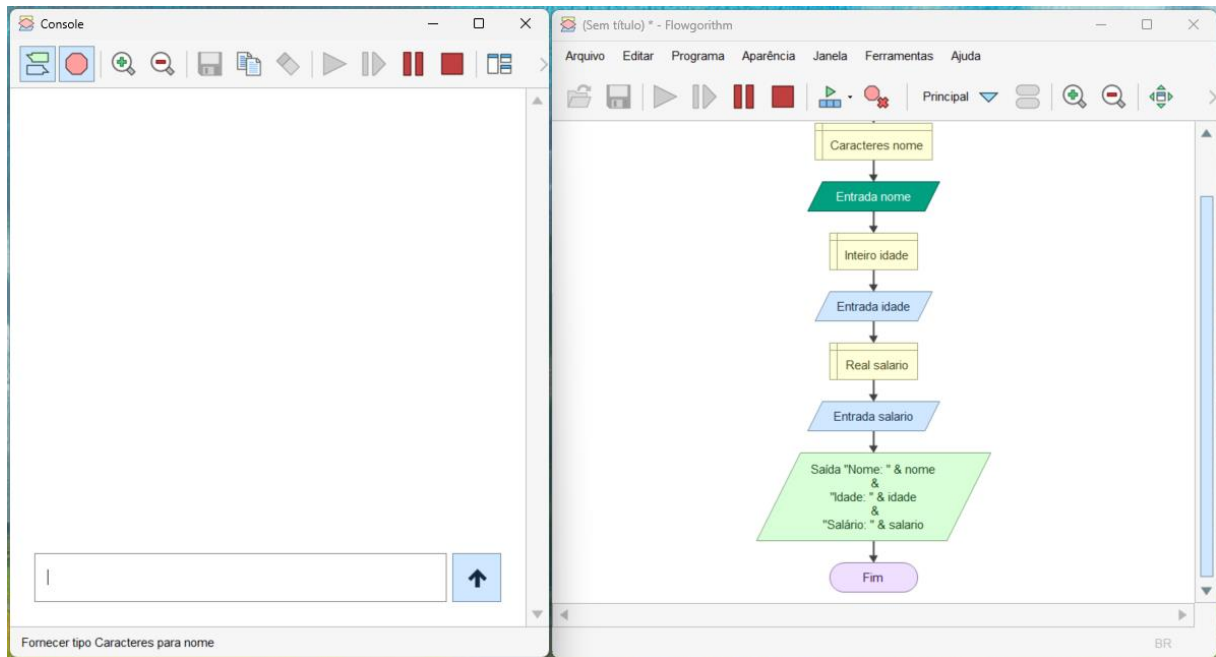
início
    escreva("Nome: ", nome, "\n");
    escreva("Idade: ", idade, "\n");
    escreva("Salário: ", salario, "\n");
fim
```



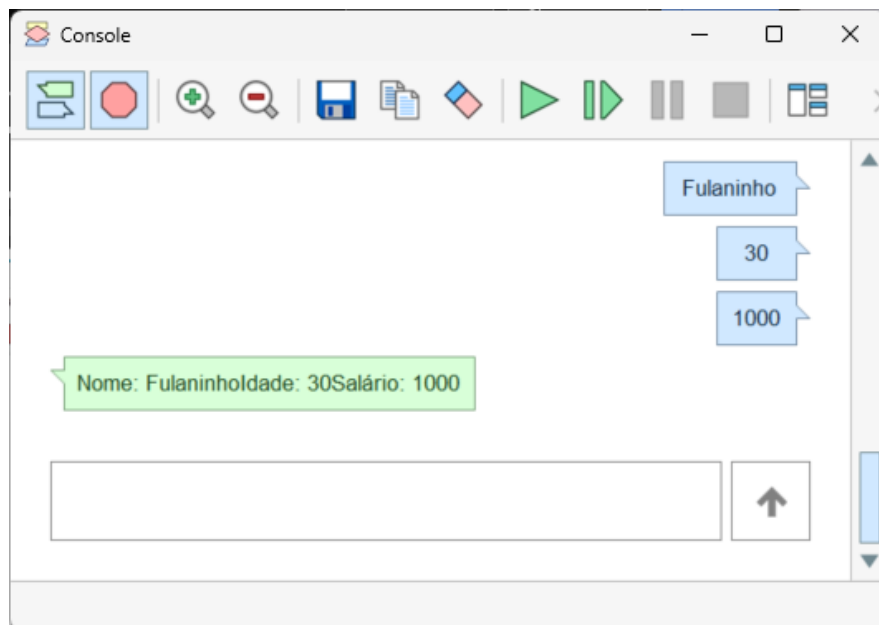
Exemplo de Entrada e Saída no Flowgorithm



Clique no botão  ou pressione F5 para executar seu código.



Informe os valores das variáveis, na ordem solicitada. Conforme mostra a figura anterior, no rodapé da janela Console, é exibida o nome da variável em leitura, e no fluxograma é possível verificar o comando que está em execução, pois está em destaque (verde escuro).



No próximo capítulo, aprenderemos sobre operadores matemáticos, relacionais e lógicos. Vamos explorar como realizar cálculos, comparações e operações lógicas em nossos algoritmos. Prepare-se para mais exemplos práticos e exercícios desafiadores!

4. Operadores Matemáticos, Relacionais e Lógicos

Neste capítulo, vamos aprender sobre os operadores matemáticos, relacionais e lógicos que são fundamentais para a manipulação de dados e a tomada de decisões em programação.

1.14. Operadores Matemáticos

Os operadores matemáticos são utilizados para realizar cálculos aritméticos.

Os principais operadores são:

- **Adição (+):** Soma dois valores.
- **Subtração (-):** Subtrai um valor do outro.
- **Multiplicação (*):** Multiplica dois valores.
- **Divisão (/):** Divide um valor pelo outro.
- **Módulo (%):** Retorna o resto da divisão de dois valores.

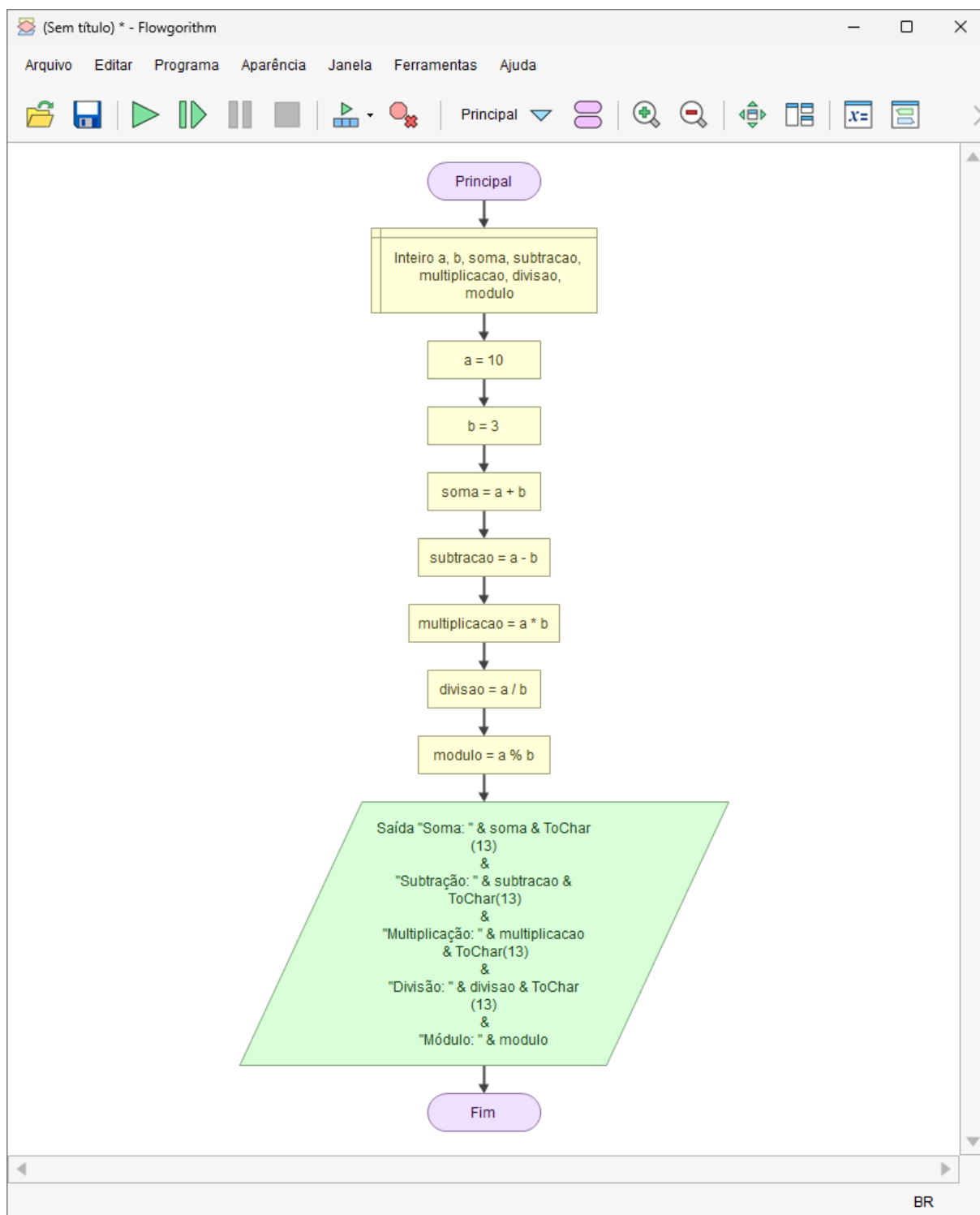
Exemplo em Portugal:

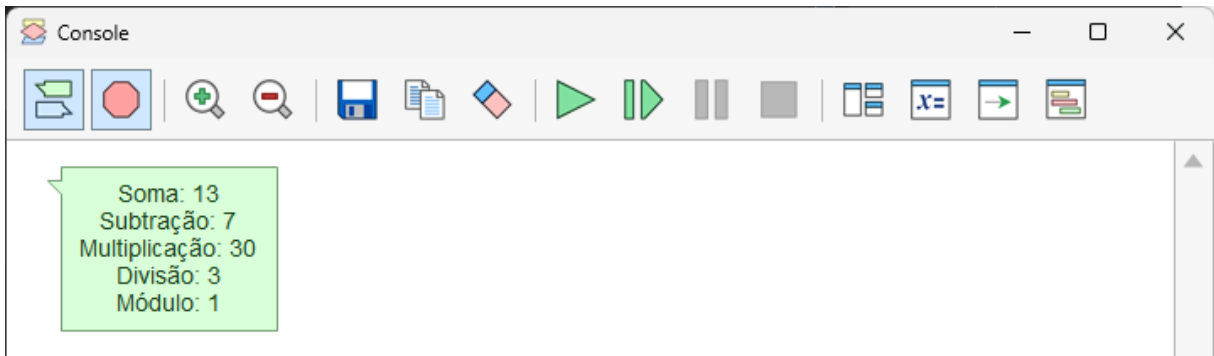
```
inteiro a = 10, b = 3;
inteiro soma, subtracao, multiplicacao, divisao, modulo;

início
    soma = a + b;           // soma = 13
    subtracao = a - b;      // subtracao = 7
    multiplicacao = a * b;  // multiplicacao = 30
    divisao = a / b;        // divisao = 3
    modulo = a % b;         // modulo = 1

    escreva("Soma: ", soma, "\n");
    escreva("Subtração: ", subtracao, "\n");
    escreva("Multiplicação: ", multiplicacao, "\n");
    escreva("Divisão: ", divisao, "\n");
    escreva("Módulo: ", modulo, "\n");
fim
```

Exemplo em Flowgorithm:





1.15. Operadores Relacionais

Os operadores relacionais são utilizados para comparar valores. Os principais operadores são:

- **Igual (==):** Verifica se dois valores são iguais.
- **Diferente (!=):** Verifica se dois valores são diferentes.
- **Maior que (>):** Verifica se um valor é maior que o outro.
- **Menor que (<):** Verifica se um valor é menor que o outro.
- **Maior ou igual (>=):** Verifica se um valor é maior ou igual ao outro.
- **Menor ou igual (<=):** Verifica se um valor é menor ou igual ao outro.

Exemplo em Portugol:

```
inteiro a = 10, b = 20;
booleano resultado;

início
    resultado = a == b; // resultado = falso
    escreva("a é igual a b? ", resultado, "\n");

    resultado = a != b; // resultado = verdadeiro
    escreva("a é diferente de b? ", resultado, "\n");

    resultado = a > b; // resultado = falso
    escreva("a é maior que b? ", resultado, "\n");

    resultado = a < b; // resultado = verdadeiro
    escreva("a é menor que b? ", resultado, "\n");

    resultado = a >= b; // resultado = falso
```

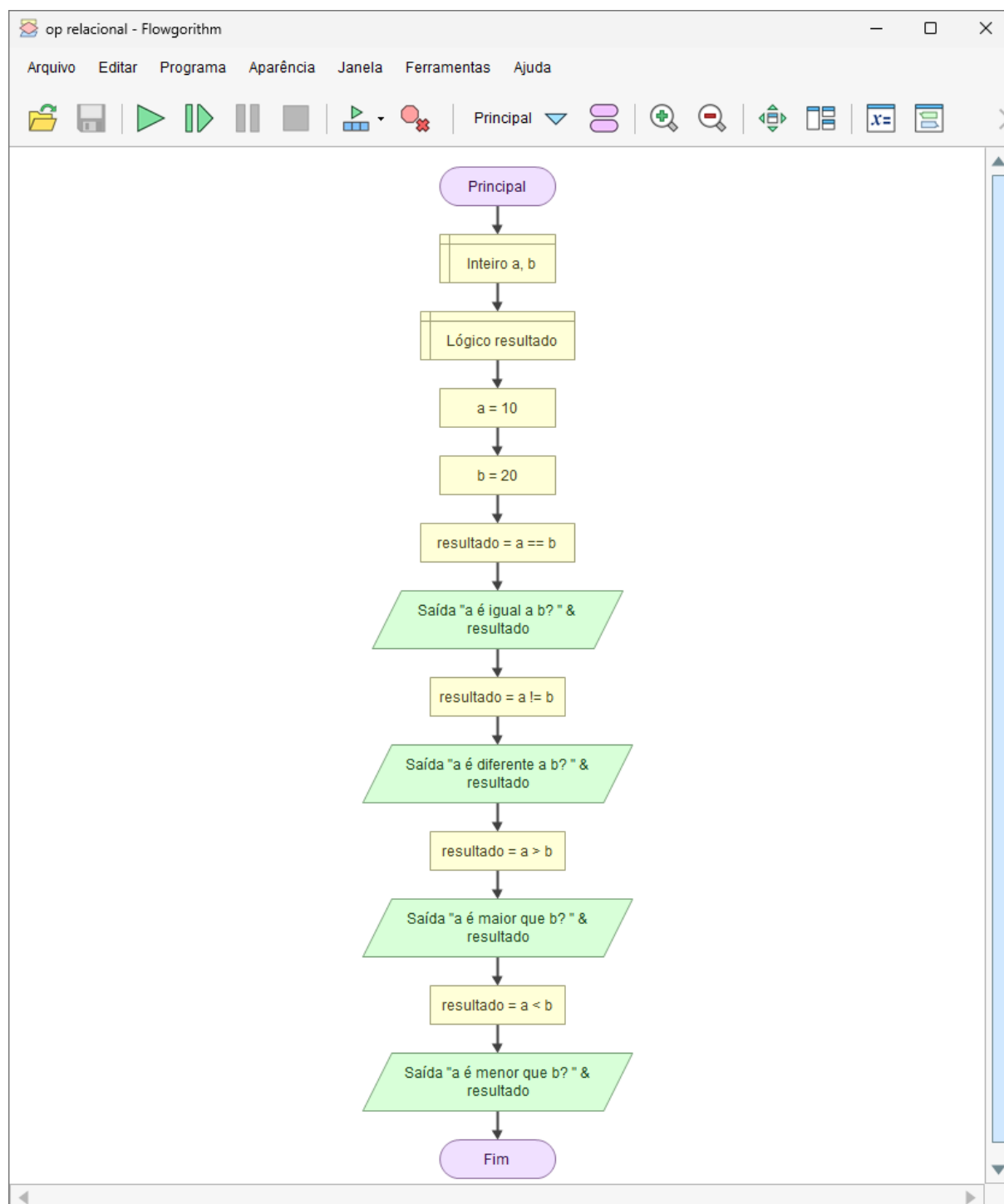
```

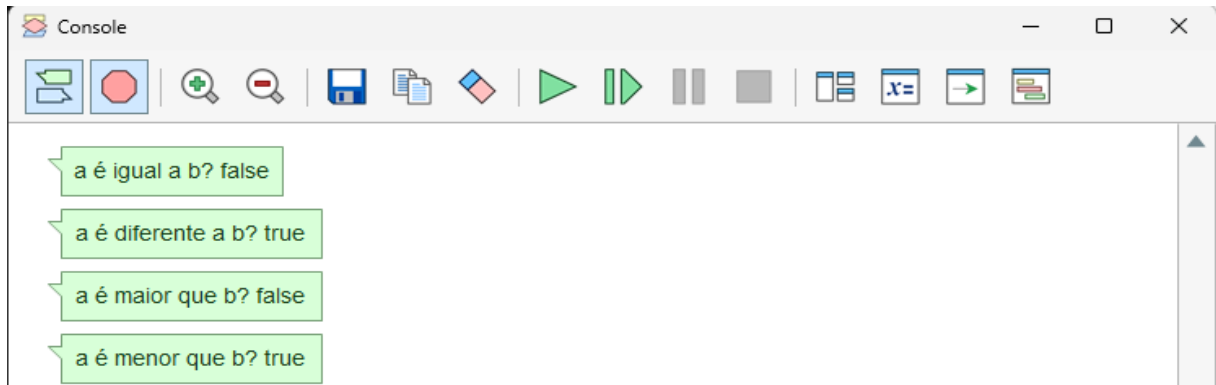
escreva("a é maior ou igual a b? ", resultado, "\n");

resultado = a <= b; // resultado = verdadeiro
escreva("a é menor ou igual a b? ", resultado, "\n");
fim

```

Exemplo em Flowgorithm:





1.16. Operadores Lógicos

Os operadores lógicos são utilizados para combinar expressões booleanas. Os principais operadores são:

- **E (&&):** Retorna verdadeiro se ambas as expressões forem verdadeiras.
- **OU (||):** Retorna verdadeiro se pelo menos uma das expressões for verdadeira.
- **NÃO (!):** Inverte o valor lógico da expressão.

Exemplo em Portugol:

```
booleano a = verdadeiro, b = falso;
booleano resultado;

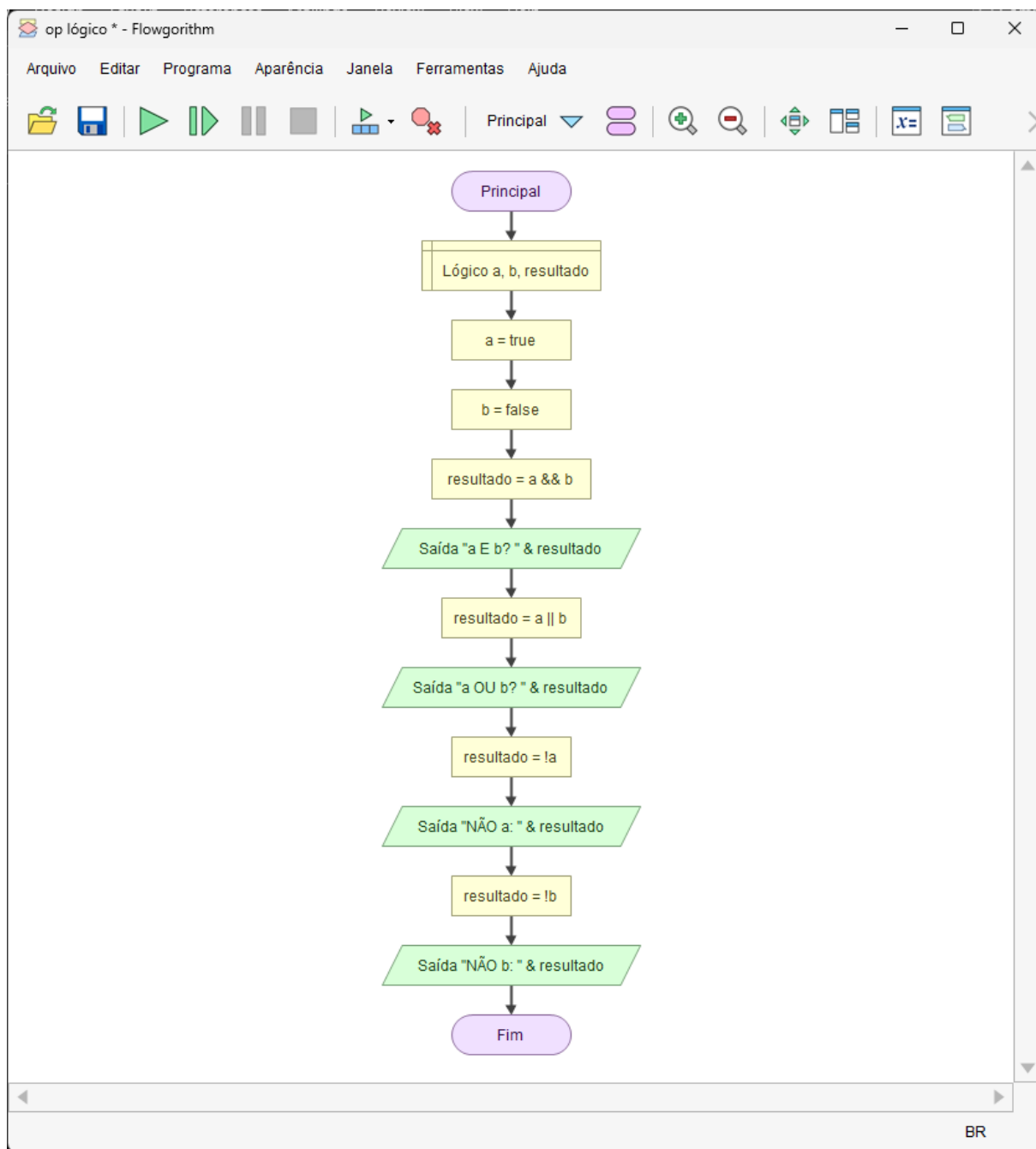
início
    resultado = a && b; // resultado = falso
    escreva("a E b? ", resultado, "\n");

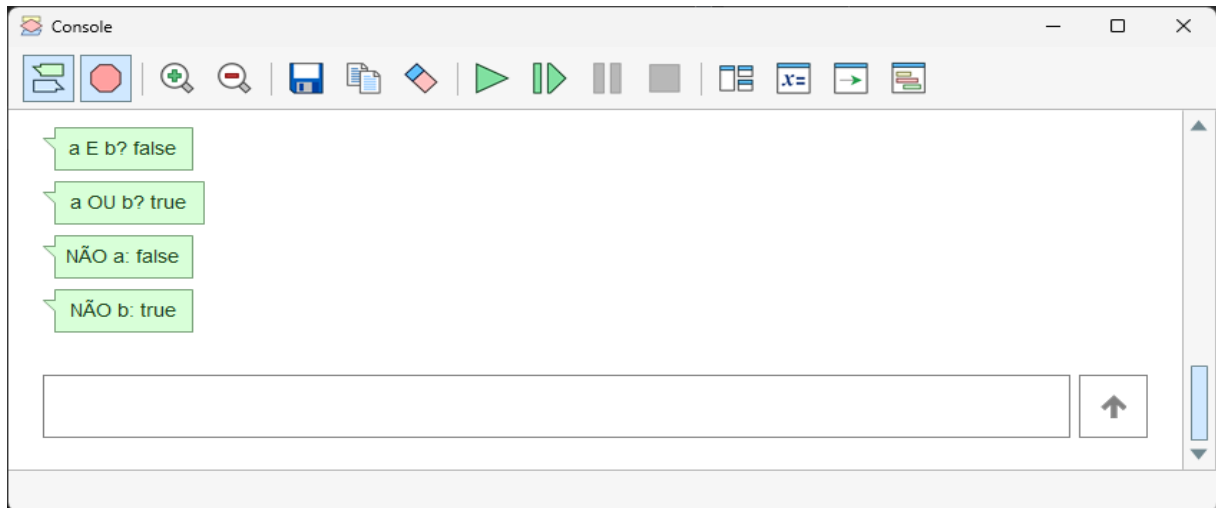
    resultado = a || b; // resultado = verdadeiro
    escreva("a OU b? ", resultado, "\n");

    resultado = !a;      // resultado = falso
    escreva("NÃO a? ", resultado, "\n");

    resultado = !b;      // resultado = verdadeiro
    escreva("NÃO b? ", resultado, "\n");
fim
```

Exemplo em Flowgorithm:





1.17. Exemplo Prático

Vamos criar um algoritmo que verifica se um número é par ou ímpar e se está dentro de um determinado intervalo.

Em português:

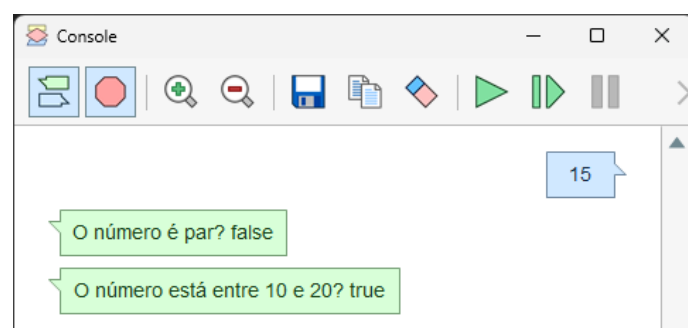
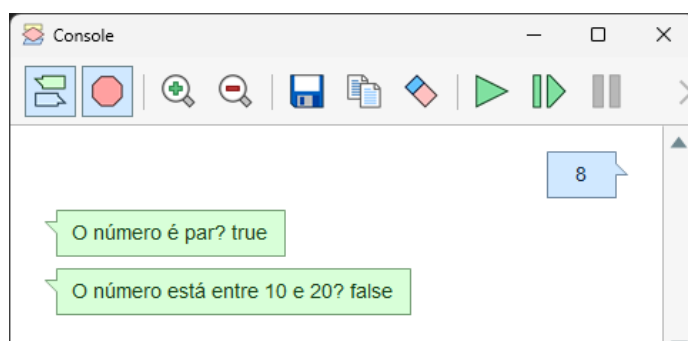
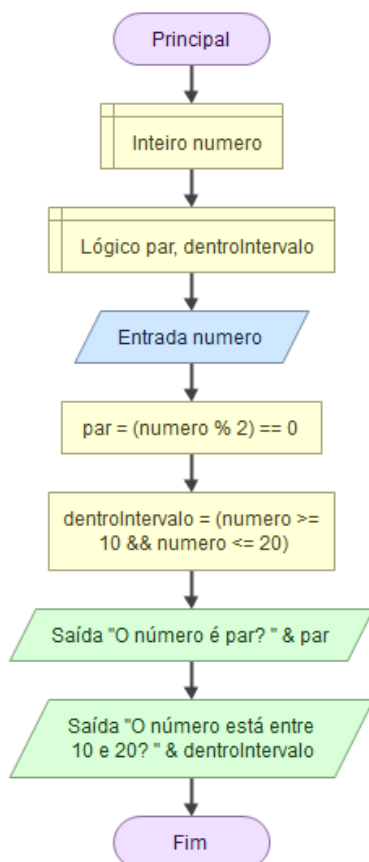
```
inteiro numero;
booleano par, dentroIntervalo;

início
    escreva("Digite um número: ");
    leia(numero);

    par = (numero % 2 == 0);
    dentroIntervalo = (numero >= 10 && numero <= 20);

    escreva("O número é par? ", par, "\n");
    escreva("O número está entre 10 e 20? ", dentroIntervalo, "\n");
fim
```


Em Flowgorithm:



5. Estruturas de Condição

Neste capítulo, vamos aprender sobre estruturas de condição, que permitem ao programa tomar decisões com base em condições específicas. Veremos como usar as instruções **"se"**, **"senão"** e **"senão se"** para controlar o fluxo do programa.

1.18. Estrutura Condicional "Se"

A estrutura condicional **"se"** (**if**) permite que o programa execute um bloco de código somente se uma condição for verdadeira.

Sintaxe:

```
se (condição) então
    // código a ser executado se a condição for verdadeira
fimse
```

Exemplo em português:

```
inteiro idade;

início
    escreva("Digite sua idade: ");
    leia(idade);

    se (idade >= 18) então
        escreva("Você é maior de idade.\n");
    fimse
fim
```

1.19. Estrutura Condicional "Senão"

A estrutura "**senão**" (**else**) é usada junto com "**se**" para executar um bloco de código alternativo se a condição for falsa.

Sintaxe:

```
se (condição) então
    // código a ser executado se a condição for verdadeira
senão
    // código a ser executado se a condição for falsa
fimse
```

Exemplo em Português:

```
inteiro idade;

início
    escreva("Digite sua idade: ");
    leia(idade);

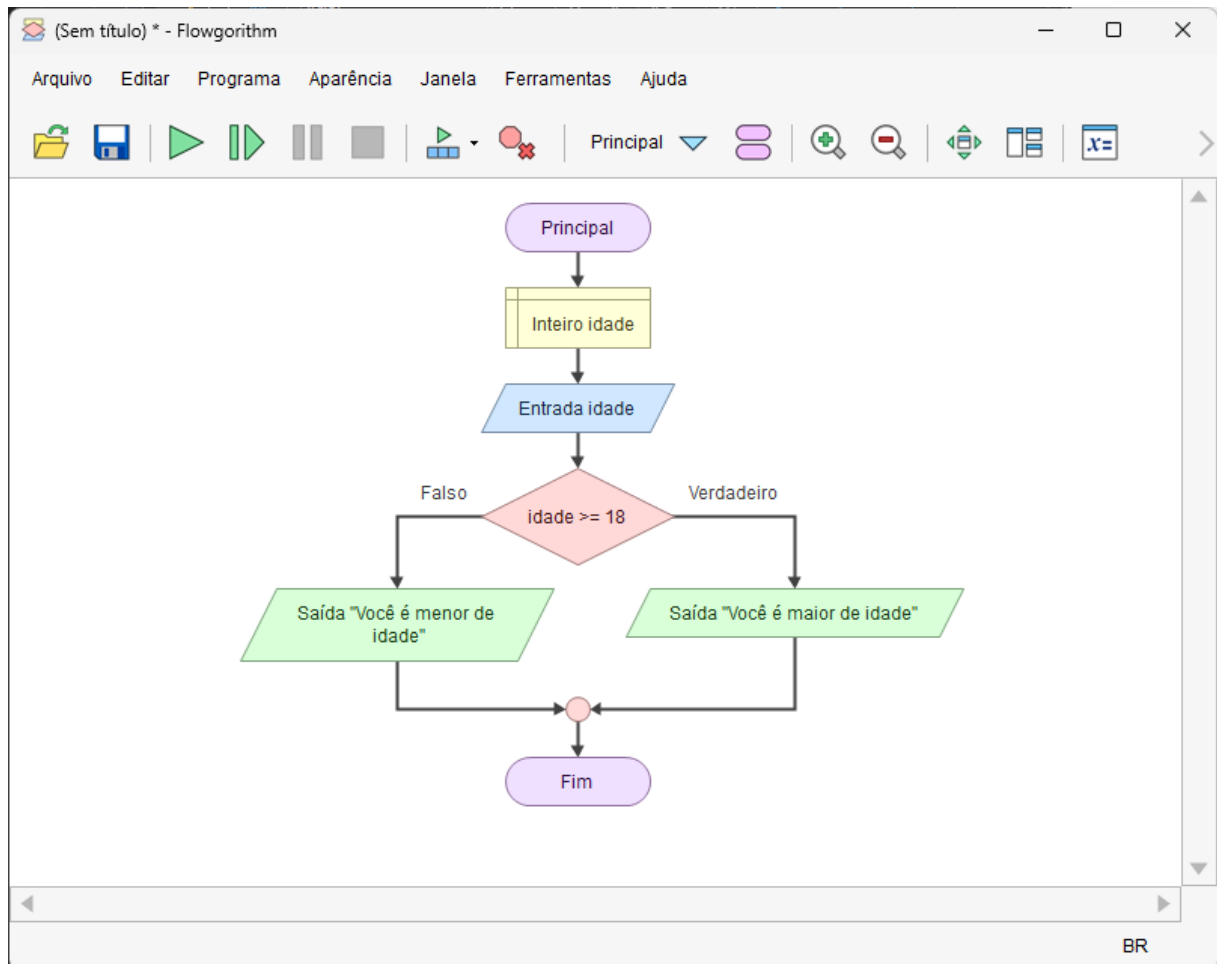
    se (idade >= 18) então
        escreva("Você é maior de idade.\n");
    senão
```

```

    escreva("Você é menor de idade.\n");
fimse
fim

```

Exemplo em Flowgorithm:



1.20. Estrutura Condicional "Senão Se"

A estrutura "**senão se**" (**else if**) permite verificar múltiplas condições em sequência.

Sintaxe:

```

se (condição1) então
    // código a ser executado se condição1 for verdadeira
senão se (condição2) então
    // código a ser executado se condição2 for verdadeira
senão
    // código a ser executado se todas as condições forem falsas
fimse

```

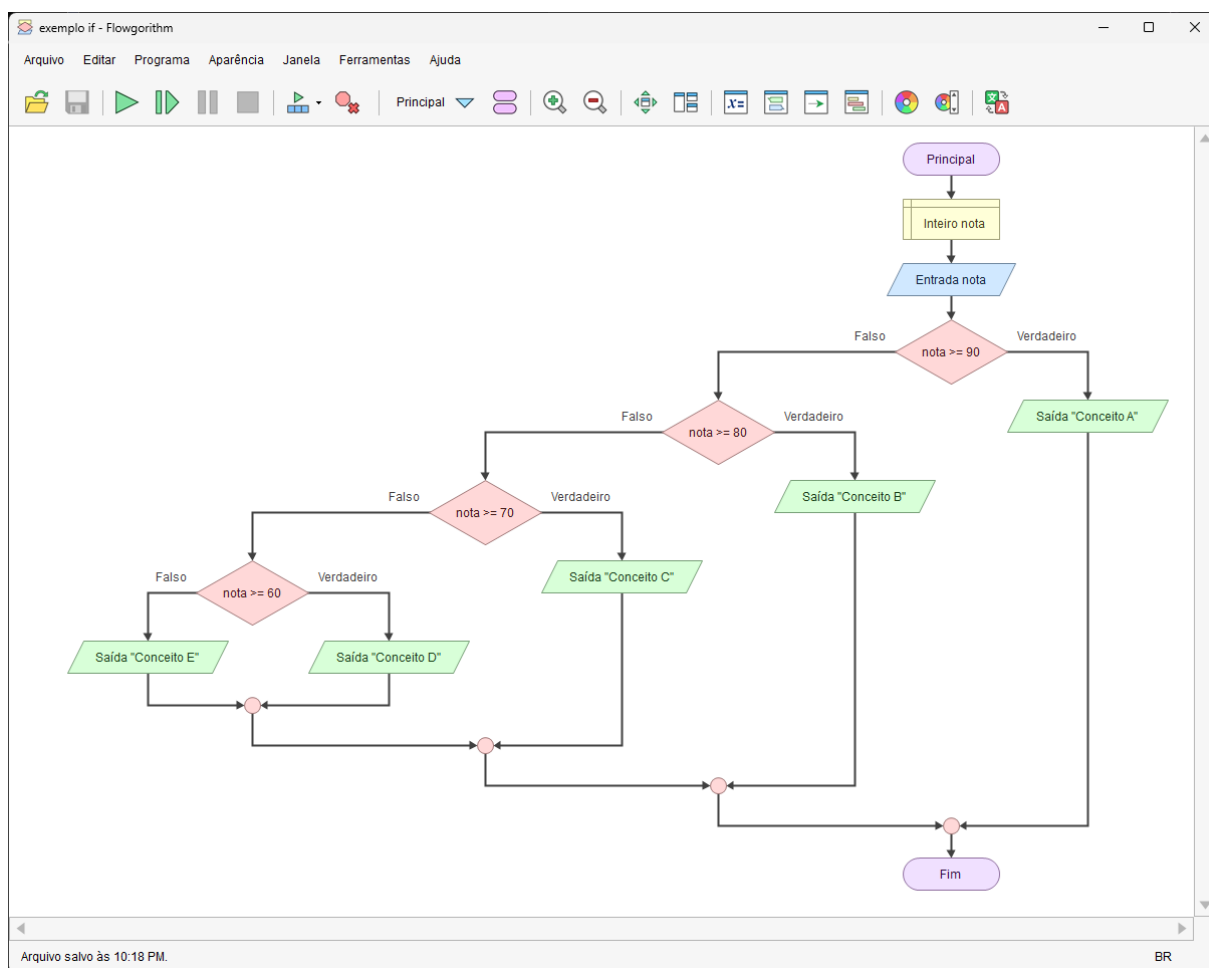
Exemplo em Portugol:

```
inteiro nota;

início
    escreva("Digite a nota do aluno: ");
    leia(nota);

    se (nota >= 90) então
        escreva("Conceito: A\n");
    senão se (nota >= 80) então
        escreva("Conceito: B\n");
    senão se (nota >= 70) então
        escreva("Conceito: C\n");
    senão se (nota >= 60) então
        escreva("Conceito: D\n");
    senão
        escreva("Conceito: F\n");
    fimse
fim
```

Exemplo em Flowgorithm:



Console window showing the execution of the flowchart for a grade of 95:

Conceito A

95

Input field:

Console window showing the execution of the flowchart for a grade of 75:

Conceito C

75

Input field:

1.21. Exemplo Prático

Vamos criar um algoritmo que determina se um número é positivo, negativo ou zero.

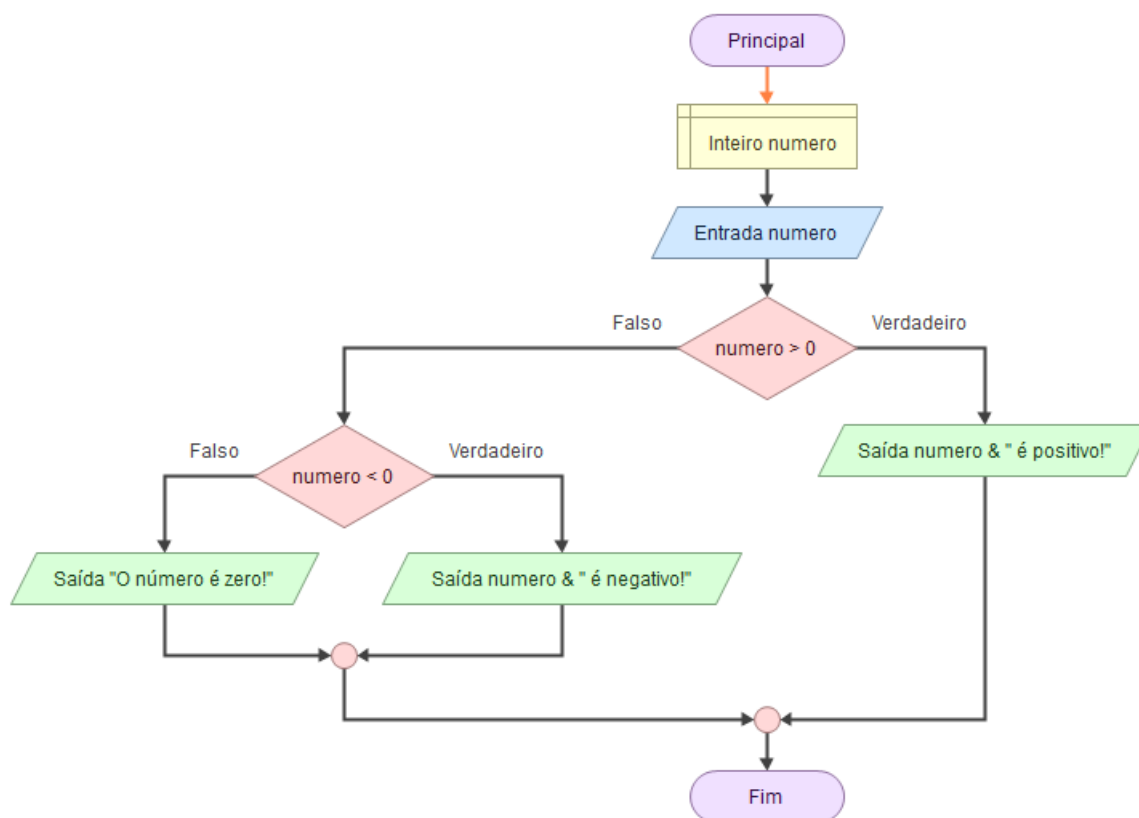
Em português:

```
inteiro numero;

início
    escreva("Digite um número: ");
    leia(numero);

    se (numero > 0) então
        escreva("O número é positivo.\n");
    senão se (numero < 0) então
        escreva("O número é negativo.\n");
    senão
        escreva("O número é zero.\n");
    fimse
fim
```

Em Flowgorithm:



6. Estruturas de Repetição

Neste capítulo, vamos aprender sobre estruturas de repetição, que permitem executar um bloco de código várias vezes. Veremos como usar os laços "para", "enquanto" e "faça enquanto" para controlar o fluxo do programa.

6.1. Estrutura de Repetição "Para"

A estrutura "**para**" (**for**) é usada quando sabemos previamente quantas vezes queremos repetir um bloco de código.

Sintaxe:

```

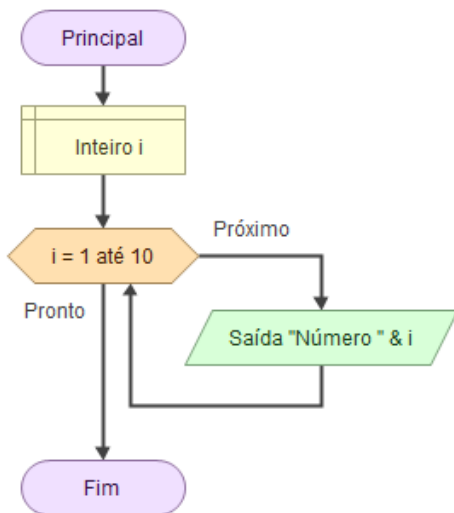
para (variável = valor_inicial; variável <= valor_final; incremento) faça
    // código a ser repetido
fimpara
  
```

Exemplo em Portugal:

```
inteiro i;

início
    para (i = 1; i <= 10; i = i + 1) faça
        escreva("Número: ", i, "\n");
    fimpara
fim
```

Exemplo em Flowgorithm:



6.2. Estrutura de Repetição "Enquanto"

A estrutura "enquanto" (while) é usada quando queremos repetir um bloco de código enquanto uma condição for verdadeira.

Sintaxe:

```
enquanto (condição) faça
    // código a ser repetido
fimenquanto
```

Exemplo em Portugal:

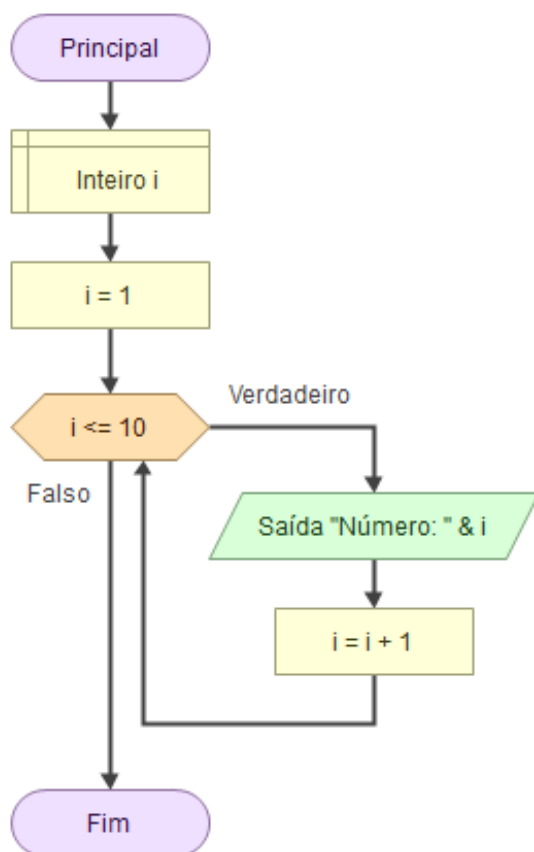
```
inteiro i = 1;

início
    enquanto (i <= 10) faça
        escreva("Número: ", i, "\n");
```



```
i = i + 1;
fimenquanto
fim
```

Exemplo em Flowgorithm:



6.3. Estrutura de Repetição "Faça Enquanto"

A estrutura **"faça enquanto"** (**do while**) é similar ao **"enquanto"**, mas a condição é verificada após a execução do bloco de código, garantindo que ele seja executado pelo menos uma vez.

Sintaxe:

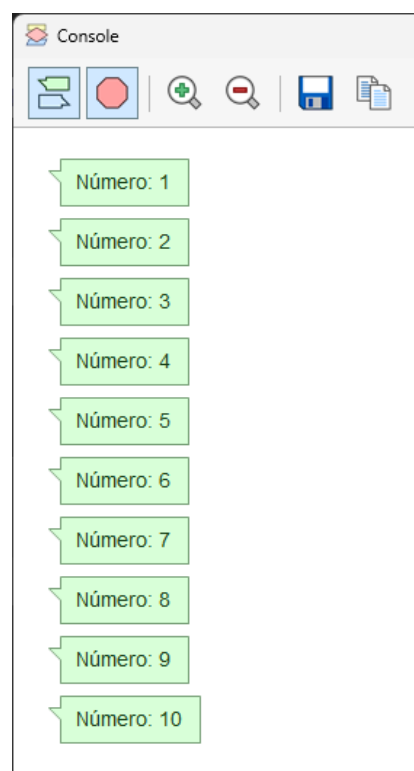
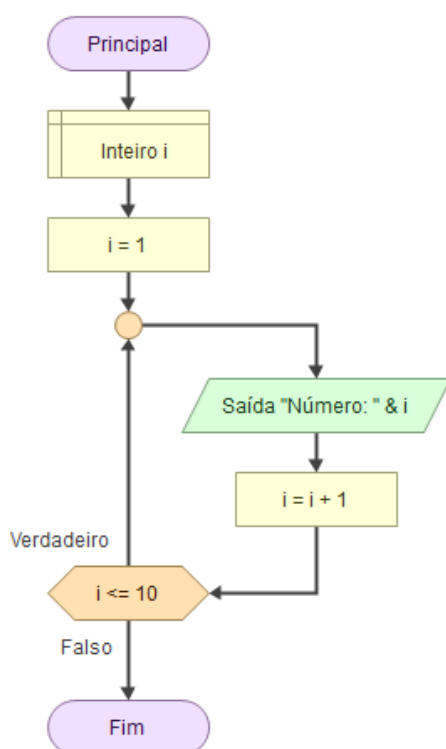
```
faça
  // código a ser repetido
enquanto (condição);
```

Exemplo:

```
inteiro i = 1;

início
    faça
        escreva("Número: ", i, "\n");
        i = i + 1;
    enquanto (i <= 10);
fim
```

Exemplo em Flowgorithm:



6.4. Exemplo Prático

Vamos criar um algoritmo que exibe os números de 1 a 10 usando cada uma das estruturas de repetição.

Algoritmo usando "Para":

```
inteiro i, numero;

início
```

```
escreva("Digite um número: ");
leia(numero);
para (i = 1; i <= 10; i = i + 1) faça
    escreva(numero, " x ", i, " = ", numero * i, "\n");
fimpara
fim
```

Algoritmo usando "Enquanto":

```
inteiro i = 1, numero;

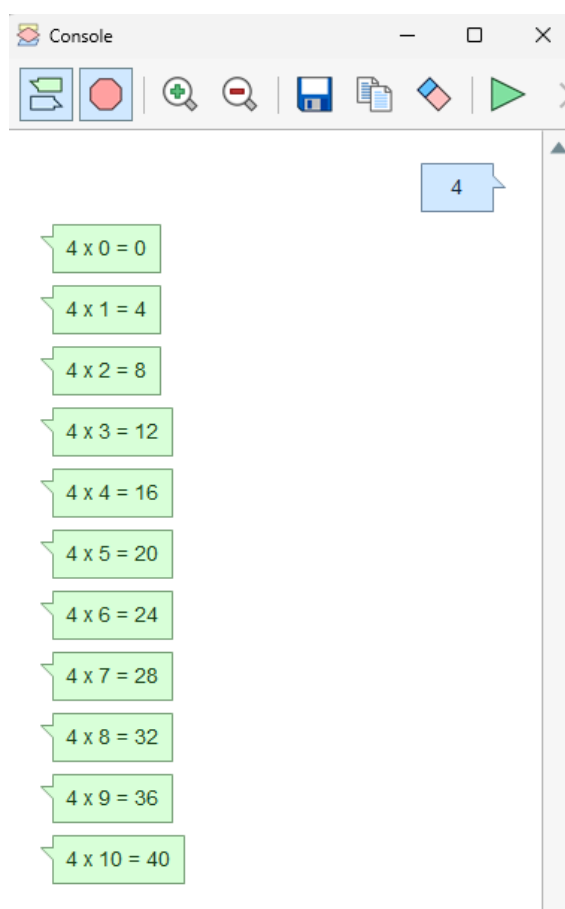
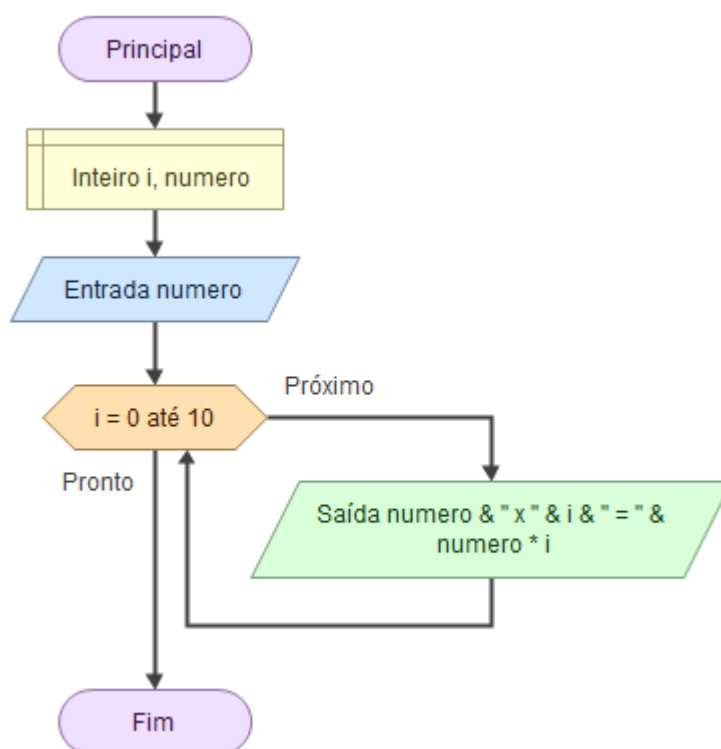
início
    escreva("Digite um número: ");
    leia(numero);
    enquanto (i <= 10) faça
        escreva(numero, " x ", i, " = ", numero * i, "\n");
        i = i + 1;
    fimenquanto
fim
```

Algoritmo usando "Faça Enquanto":

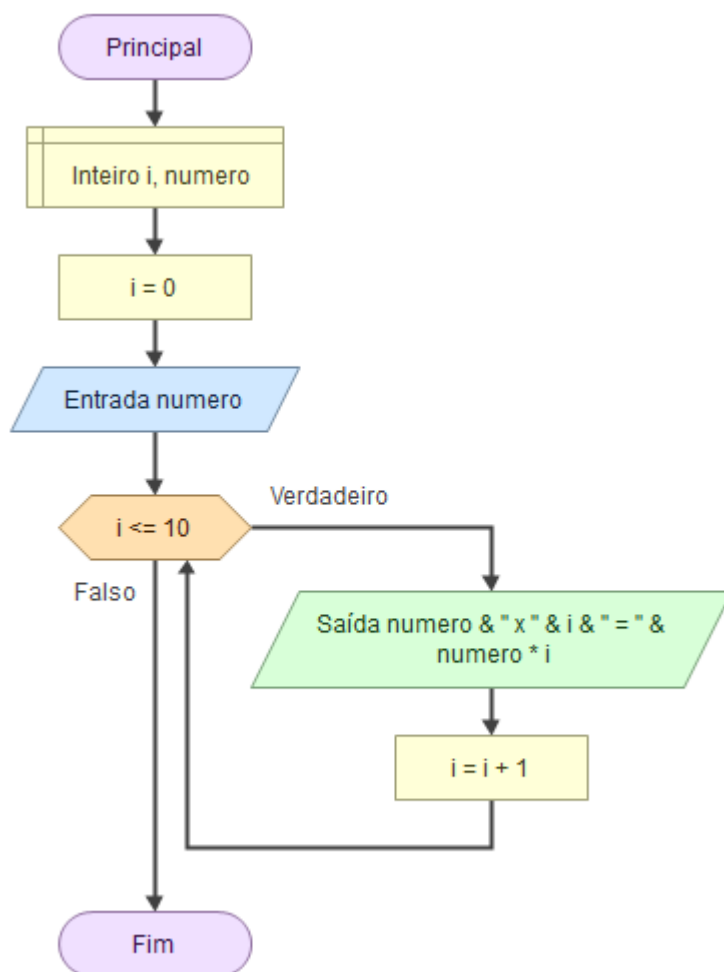
```
inteiro i = 1, numero;

início
    escreva("Digite um número: ");
    leia(numero);
    faça
        escreva(numero, " x ", i, " = ", numero * i, "\n");
        i = i + 1;
    enquanto (i <= 10);
fim
```

Flowgorithm com “Para”



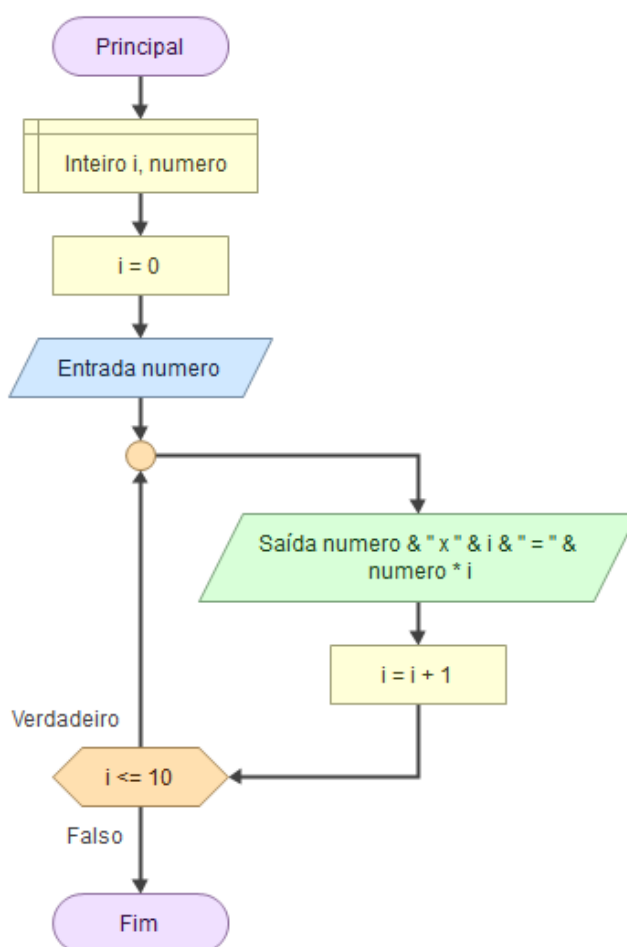
Flowgorithm com “Enquanto”

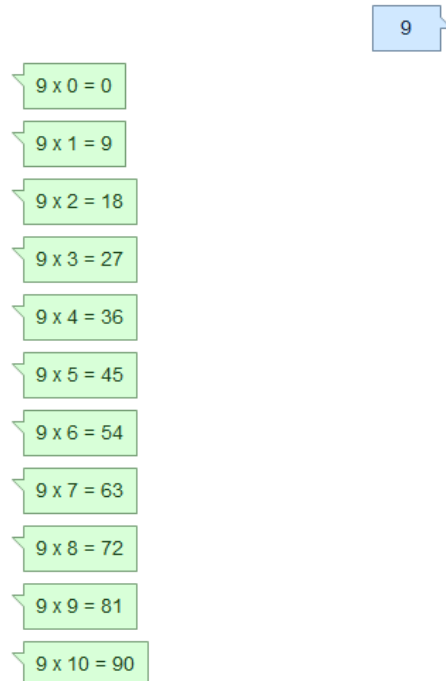


6

6 x 0 = 0
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60

Flowgorithm com “Faça Enquanto”





Vetores e Matrizes

Neste capítulo, vamos aprender sobre vetores e matrizes, que são estruturas de dados que nos permitem armazenar e manipular coleções de valores. Veremos como declarar, inicializar e acessar elementos dessas estruturas.

6.1. Vetores

Um vetor é uma coleção de elementos do mesmo tipo, armazenados em posições consecutivas na memória. Cada elemento pode ser acessado através de um índice.

Declaração de Vetores:

```
inteiro vetor[10]; // Declara um vetor de 10 inteiros
```

Inicialização de Vetores:

```
vetor[0] = 1;
vetor[1] = 2;
vetor[2] = 3;
// ...
```

Exemplo em Portugol:

```

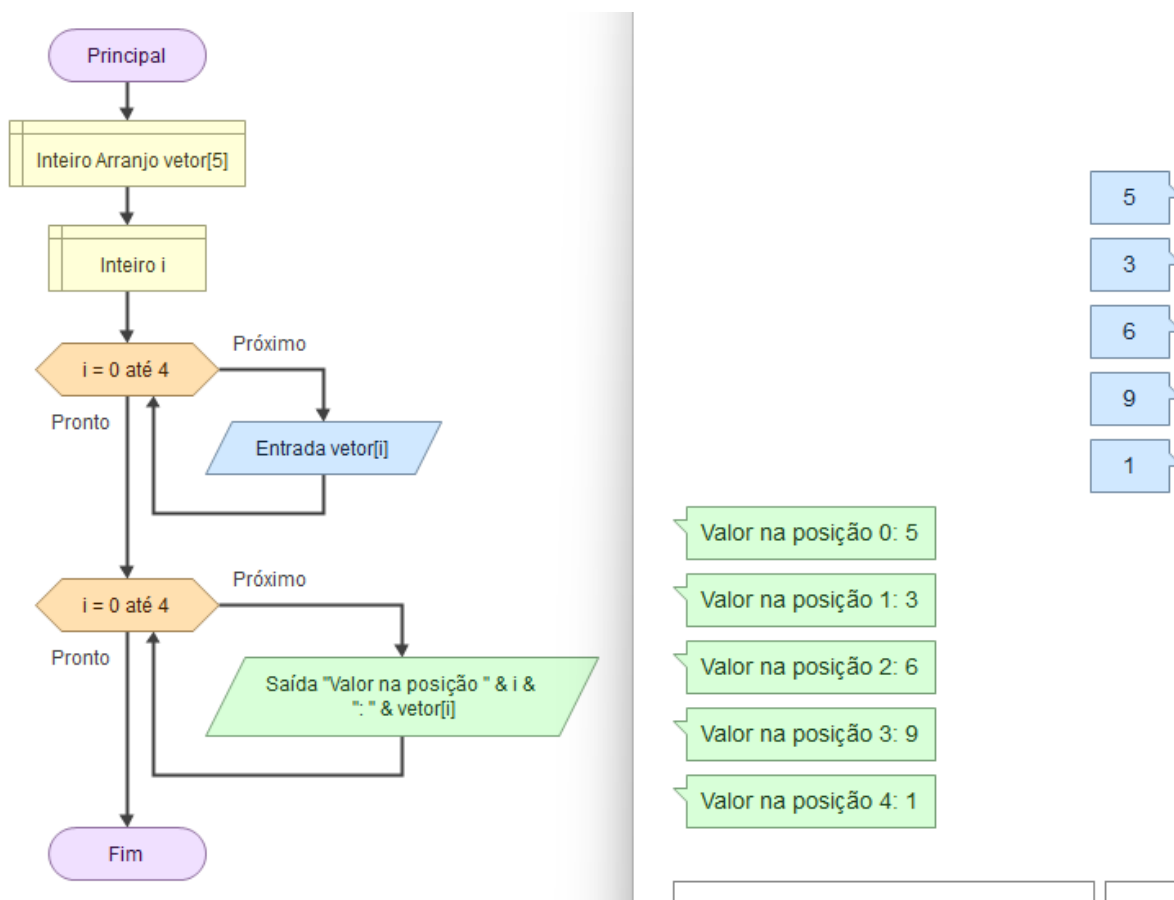
inteiro vetor[5];
inteiro i;

início
    // Inicializando o vetor
    para (i = 0; i < 5; i = i + 1) faça
        escreva("Digite o valor para a posição ", i, ": ");
        leia(vetor[i]);
    fimpara

    // Exibindo os valores do vetor
    para (i = 0; i < 5; i = i + 1) faça
        escreva("Valor na posição ", i, ": ", vetor[i], "\n");
    fimpara
fim

```

Exemplo em Flowgorithm:



6.2. Matrizes

Uma matriz é uma coleção de elementos do mesmo tipo organizados em linhas e colunas. Cada elemento pode ser acessado através de dois índices: um para a linha e outro para a coluna.

Declaração de Matrizes:

```
inteiro matriz[3][3]; // Declara uma matriz 3x3 de inteiros
```

Inicialização de Matrizes:

```
matriz[0][0] = 1;  
matriz[0][1] = 2;  
matriz[0][2] = 3;  
// ...
```

Exemplo em Portugol:

```
inteiro matriz[2][2];  
inteiro i, j;  
  
início  
    // Inicializando a matriz  
    para (i = 0; i < 2; i = i + 1) faça  
        para (j = 0; j < 2; j = j + 1) faça  
            escreva("Digite o valor para a posição [", i, "][", j, "]: ");  
            leia(matriz[i][j]);  
        fimpara  
    fimpara  
  
    // Exibindo os valores da matriz  
    para (i = 0; i < 2; i = i + 1) faça  
        para (j = 0; j < 2; j = j + 1) faça  
            escreva("Valor na posição [", i, "][", j, "]: ", matriz[i][j], "\n");  
        fimpara  
    fimpara  
fim
```

No momento não é possível utilizar matrizes no flowgorithm

7. Funções e Procedimentos

Neste capítulo, vamos aprender sobre funções e procedimentos, que são sub-rotinas que nos permitem organizar e reutilizar código. Veremos como declarar, definir e chamar essas sub-rotinas.

7.1. Funções

Funções são sub-rotinas que realizam uma tarefa específica e retornam um valor. Elas ajudam a modularizar o código e facilitar a manutenção.


Declaração de Funções:

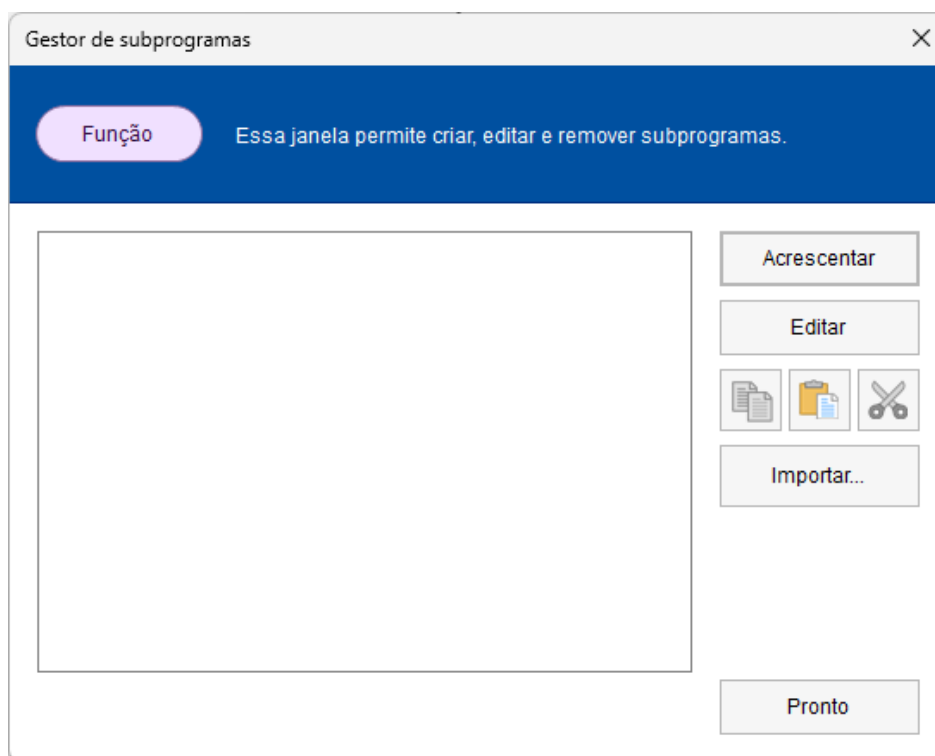
```
tipo nome_da_funcao(parâmetros) {  
    // corpo da função  
    retorne valor;  
}
```

Exemplo de Função no Portugol:

```
inteiro soma(inteiro a, inteiro b) {  
    inteiro resultado;  
    resultado = a + b;  
    retorne resultado;  
}  
  
início  
    inteiro x = 5, y = 10, z;  
    z = soma(x, y);  
    escreva("A soma de ", x, " e ", y, " é: ", z, "\n");  
fim
```

Exemplo de Função no Flowgorithm:

Primeiro criamos a função, através do botão  Gestor de subprogramas.



Função - Propriedades

Função

Uma função permite ao programa a reutilização de código e a simplificação da lógica. Dados são passados às funções através de parâmetros.

Nome do subprograma:

soma

Parâmetros:

Inteiro a

Inteiro b

↑

↓

Acrescentar

Editar

Remover

Tipo do retorno:

☒ Inteiro

☐ Real

☐ Caracteres

☐ Lógico

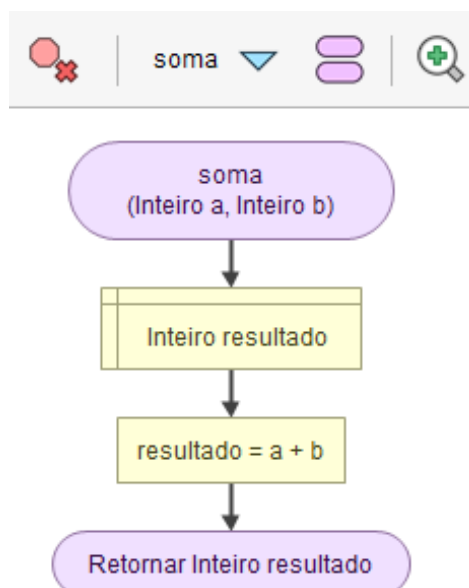
☐ Nenhum

Variável de retorno:

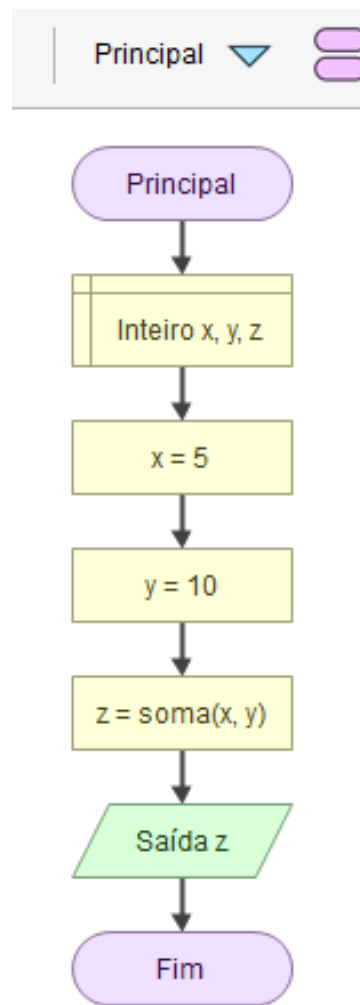
resultado

OK

Cancelar



Agora basta voltar ao programa **Principal** e usar a função:



7.2. Procedimentos

Procedimentos são sub-rotinas que realizam uma tarefa específica, mas não retornam um valor. Eles também ajudam a modularizar o código e facilitar a manutenção.

Declaração de Procedimentos:

```

procedimento nome_do_procedimento(parâmetros) {
    // corpo do procedimento
}
  
```

Exemplo de Procedimento em Portugol:

```

procedimento saudacao(string nome) {
    escreva("Olá, ", nome, "!\n");
}
  
```

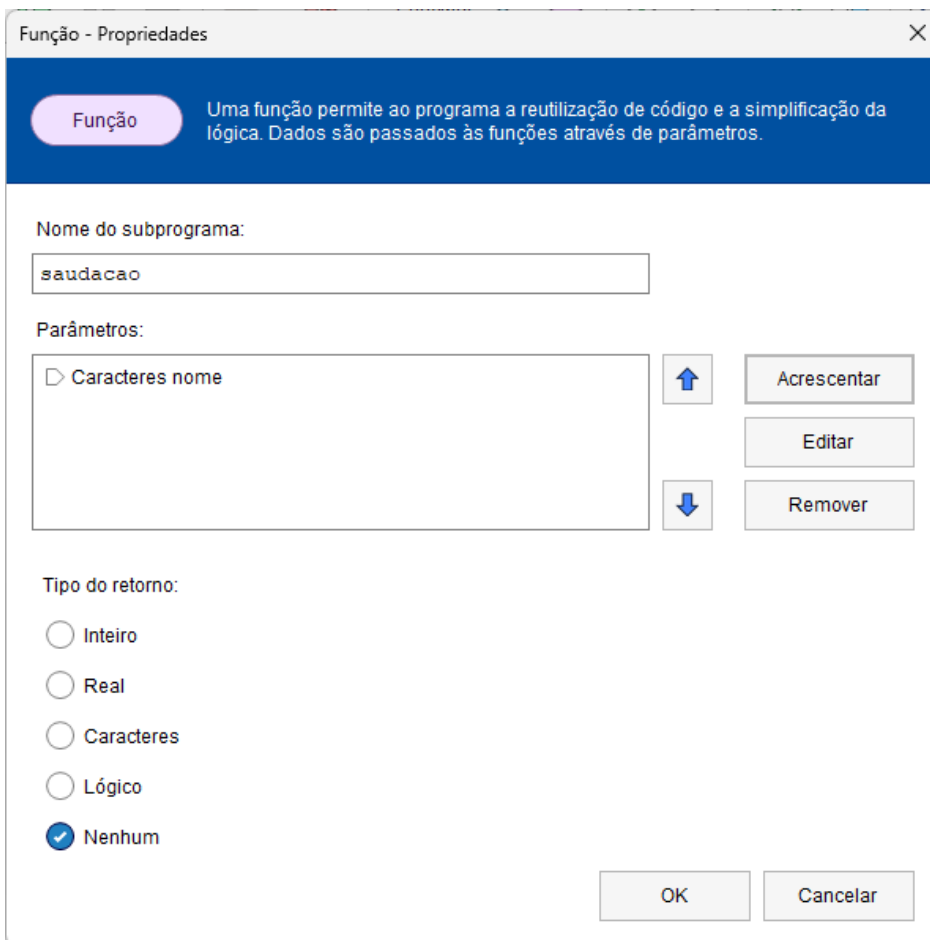
```

}

início
    saudacao("João");
fim

```

Exemplo de Procedimento no Flowgorithm:



Função - Propriedades

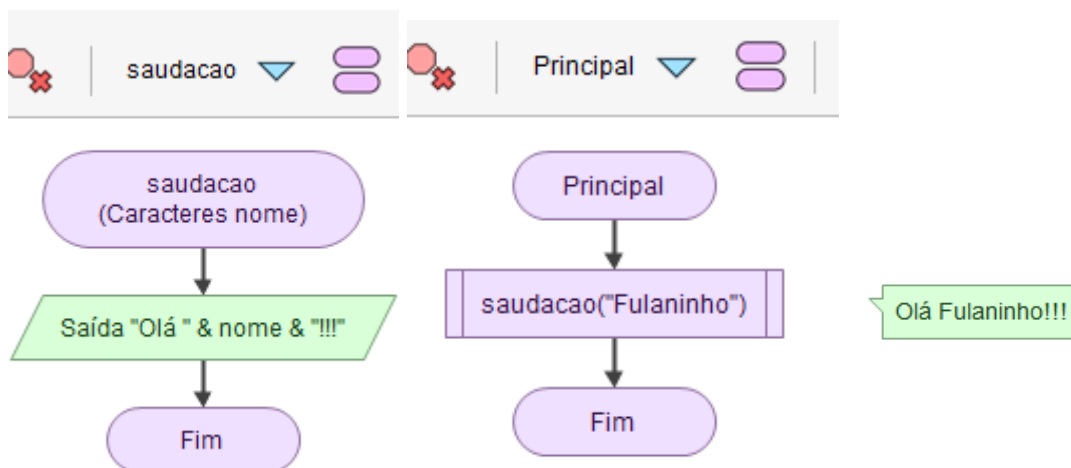
Função Uma função permite ao programa a reutilização de código e a simplificação da lógica. Dados são passados às funções através de parâmetros.

Nome do subprograma:
saudacao

Parâmetros:
☐ Caracteres nome

Tipo do retorno:

☐ Inteiro
☐ Real
☐ Caracteres
☐ Lógico
☒ Nenhum



7.3. Diferenças entre Funções e Procedimentos

- Funções retornam um valor.
- Procedimentos não retornam valor.

Exercícios Práticos

Com base nos conceitos estudados, desenvolva os fluxogramas necessários a resolução das solicitações abaixo, de acordo com o material estudado. A resolução de cada exercício encontra-se ao final deste material

Variáveis, Constantes e Tipos de Dados

1. Crie um algoritmo que leia dois números inteiros e exiba a soma, subtração, multiplicação e divisão deles.
2. Faça um algoritmo que leia o nome, a idade e o salário de uma pessoa e exiba essas informações formatadas.

Entrada e Saída de Dados

1. Crie um algoritmo que leia três números inteiros e exiba a média deles.
2. Faça um algoritmo que leia o nome e a altura de uma pessoa e exiba uma mensagem dizendo "Nome, você tem X metros de altura".
3. Desenvolva um algoritmo que leia um valor em reais e a cotação do dólar, e exiba o valor convertido para dólares.

Operadores Matemáticos, Relacionais e Lógicos

1. Crie um algoritmo que leia dois números e exiba se o primeiro é maior, menor ou igual ao segundo.
2. Faça um algoritmo que leia três números e exiba se todos são iguais.

Estruturas de Condição

1. Desenvolva um algoritmo que leia a idade de uma pessoa e determine se ela é maior de idade ($\text{idade} \geq 18$).
2. Crie um algoritmo que leia a nota de um aluno e exiba se ele foi aprovado ($\text{nota} \geq 6$) ou reprovado ($\text{nota} < 6$).
3. Faça um algoritmo que leia três números e exiba o maior deles.
4. Desenvolva um algoritmo que calcule a média de três notas e determine se o aluno está aprovado ($\text{média} \geq 6$) ou reprovado ($\text{média} < 6$).
5. Desenvolva um algoritmo que leia a temperatura em graus Celsius e determine se está frio (abaixo de 15), ameno (entre 15 e 25) ou quente (acima de 25).

Estruturas de Repetição

1. Crie um algoritmo que exiba todos os números pares de 1 a 20 usando a estrutura "para".
2. Faça um algoritmo que leia 5 números e exiba a soma deles usando a estrutura "enquanto".
3. Desenvolva um algoritmo que leia números do usuário até que ele digite um número negativo, e então exiba a média dos números positivos digitados usando a estrutura "faça enquanto".

Vetores e Matrizes

1. Crie um algoritmo que leia 10 números inteiros e exiba a média deles.
2. Desenvolva um algoritmo que leia um vetor de 5 posições e exiba os elementos em ordem inversa.

Funções e Procedimentos

1. Crie uma função que receba um número inteiro e retorne seu fatorial.
2. Faça um procedimento que receba um vetor de 10 inteiros e exiba os valores na tela.
3. Desenvolva uma função que receba um vetor de 5 reais e retorne a soma dos elementos.

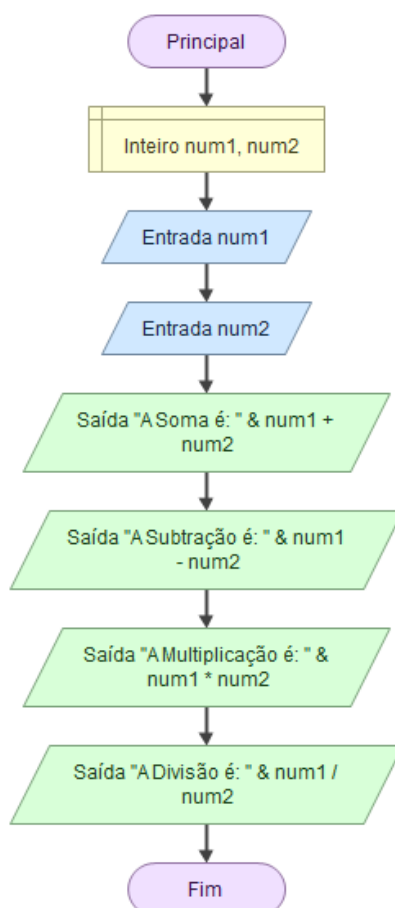
Variáveis, Constantes e Tipos de Dados

1. Crie um algoritmo que leia dois números inteiros e exiba a soma, subtração, multiplicação e divisão deles.

```
programa
{
    funcao inicio()
    {
        inteiro num1, num2;

        escreva("Digite o primeiro número: ");
        leia(num1);
        escreva("Digite o segundo número: ");
        leia(num2);

        escreva("Soma: ", num1 + num2, "\n");
        escreva("Subtração: ", num1 - num2, "\n");
        escreva("Multiplicação: ", num1 * num2, "\n");
        escreva("Divisão: ", num1 / num2, "\n");
    }
}
```



Variáveis, Constantes e Tipos de Dados

2. Faça um algoritmo que leia o nome, a idade e o salário de uma pessoa e exiba essas informações formatadas.

```

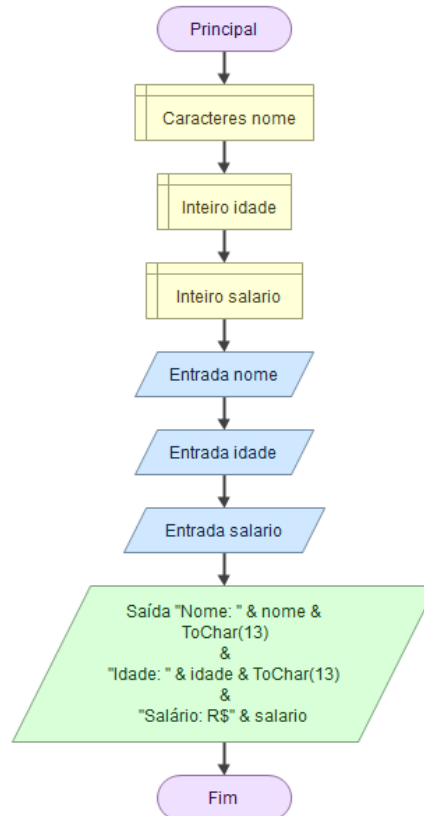
programa
{
    funcao inicio()
    {
        caractere nome[50];
        inteiro idade;
        real salario;

        escreva("Digite seu nome: ");
        leia(nome);
        escreva("Digite sua idade: ");
        leia(idade);
        escreva("Digite seu salário: ");
        leia(salario);
    }
}
  
```

```

    escreva("Nome: ", nome, "\n");
    escreva("Idade: ", idade, " anos\n");
    escreva("Salário: R$", salario, "\n");
  }
}

```



Entrada e Saída de Dados

1. Crie um algoritmo que leia três números inteiros e exiba a média deles.

```

programa
{
    funcao inicio()
    {
        inteiro num1, num2, num3;
        real media;

        escreva("Digite o primeiro número: ");
        leia(num1);
        escreva("Digite o segundo número: ");
        leia(num2);
        escreva("Digite o terceiro número: ");
        leia(num3);
    }
}

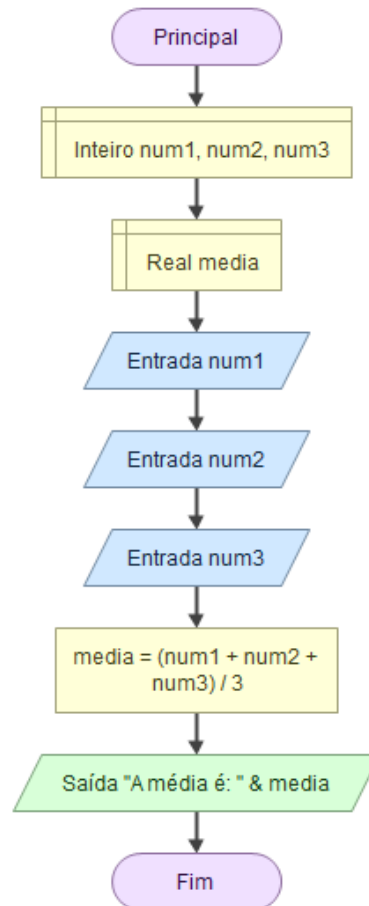
```

```

media = (num1 + num2 + num3) / 3;

escreva("A média é: ", media, "\n");
}
}

```



Entrada e Saída de Dados

2. Faça um algoritmo que leia o nome e a altura de uma pessoa e exiba uma mensagem dizendo "Nome, você tem X metros de altura".

```

programa
{
    funcao inicio()
    {
        caractere nome[50];
        real altura;

        escreva("Digite seu nome: ");
        leia(nome);
        escreva("Digite sua altura (em metros): ");
    }
}

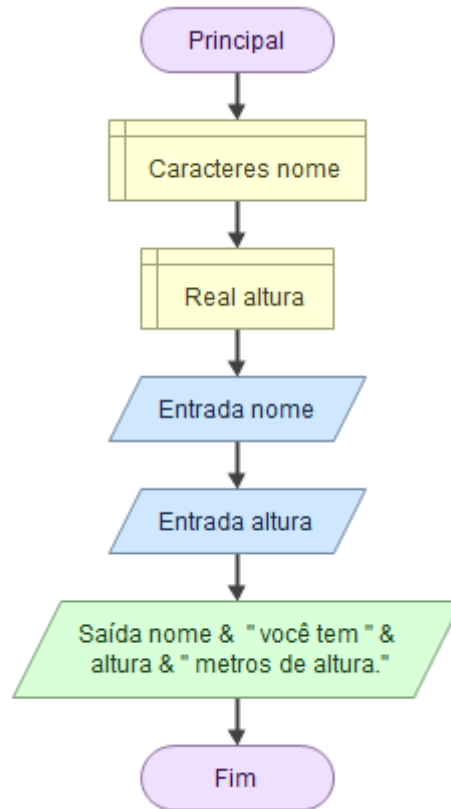
```

```

leia(altura);

escreva(nome, ", você tem ", altura, " metros de altura.\n");
}
}

```



Entrada e Saída de Dados

- Desenvolva um algoritmo que leia um valor em reais e a cotação do dólar, e exiba o valor convertido para dólares.

```

programa
{
    funcao inicio()
    {
        real reais, cotacao, dolares;

        escreva("Digite o valor em reais: ");
        leia(reais);
        escreva("Digite a cotação do dólar: ");
        leia(cotacao);

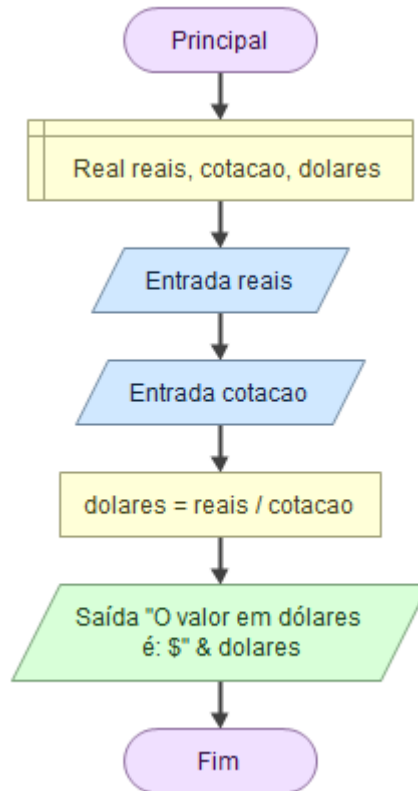
        dolares = reais / cotacao;
    }
}

```

```

    escreva("O valor em dólares é: $", dolares, "\n");
  }
}

```



Operadores Matemáticos, Relacionais e Lógicos

1. Crie um algoritmo que leia dois números e exiba se o primeiro é maior, menor ou igual ao segundo.

```

programa
{
  funcao inicio()
  {
    inteiro num1, num2;

    escreva("Digite o primeiro número: ");
    leia(num1);
    escreva("Digite o segundo número: ");
    leia(num2);

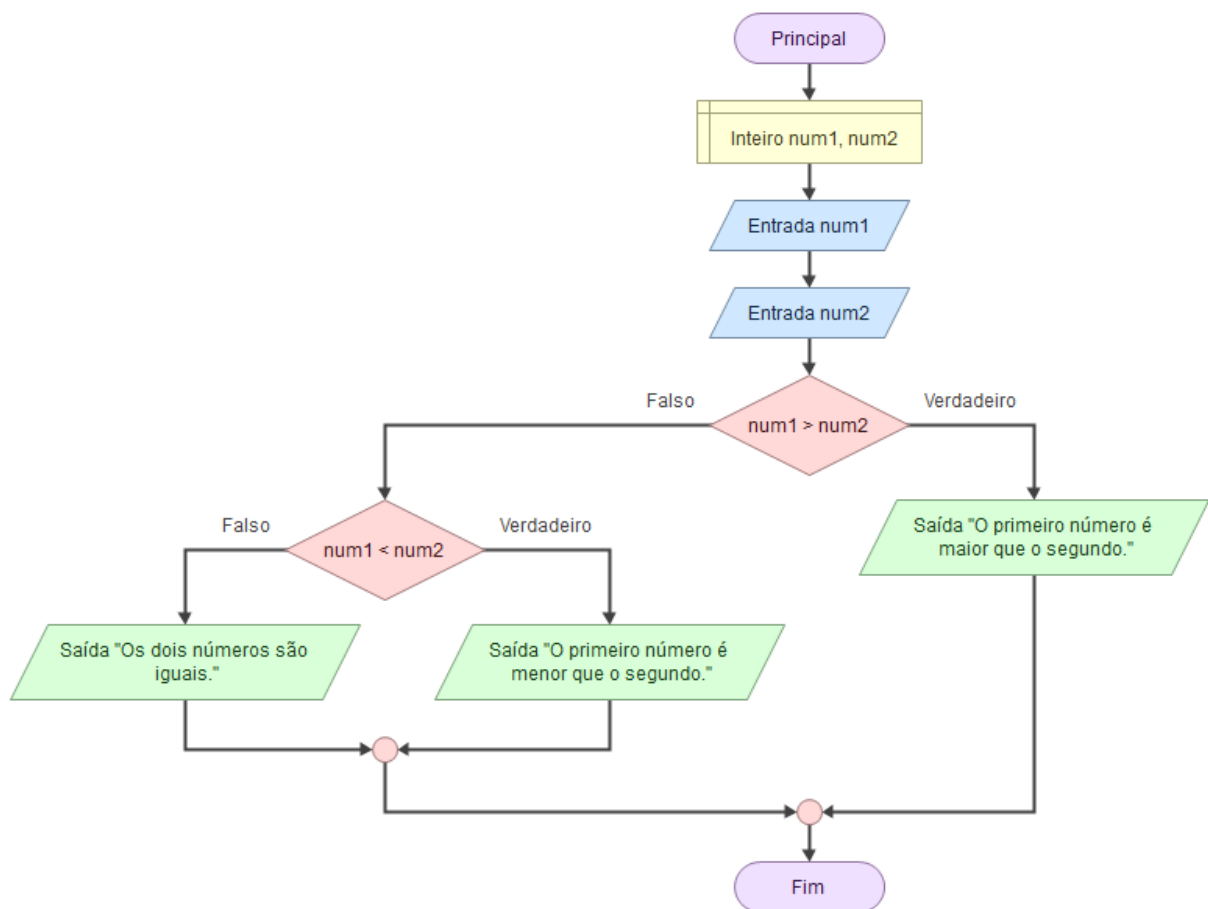
    se (num1 > num2)
    {
      escreva("O primeiro número é maior que o segundo.\n");
    }
  }
}

```

```

    }
    senao se (num1 < num2)
    {
        escreva("O primeiro número é menor que o segundo.\n");
    }
    senao
    {
        escreva("Os dois números são iguais.\n");
    }
}
}

```



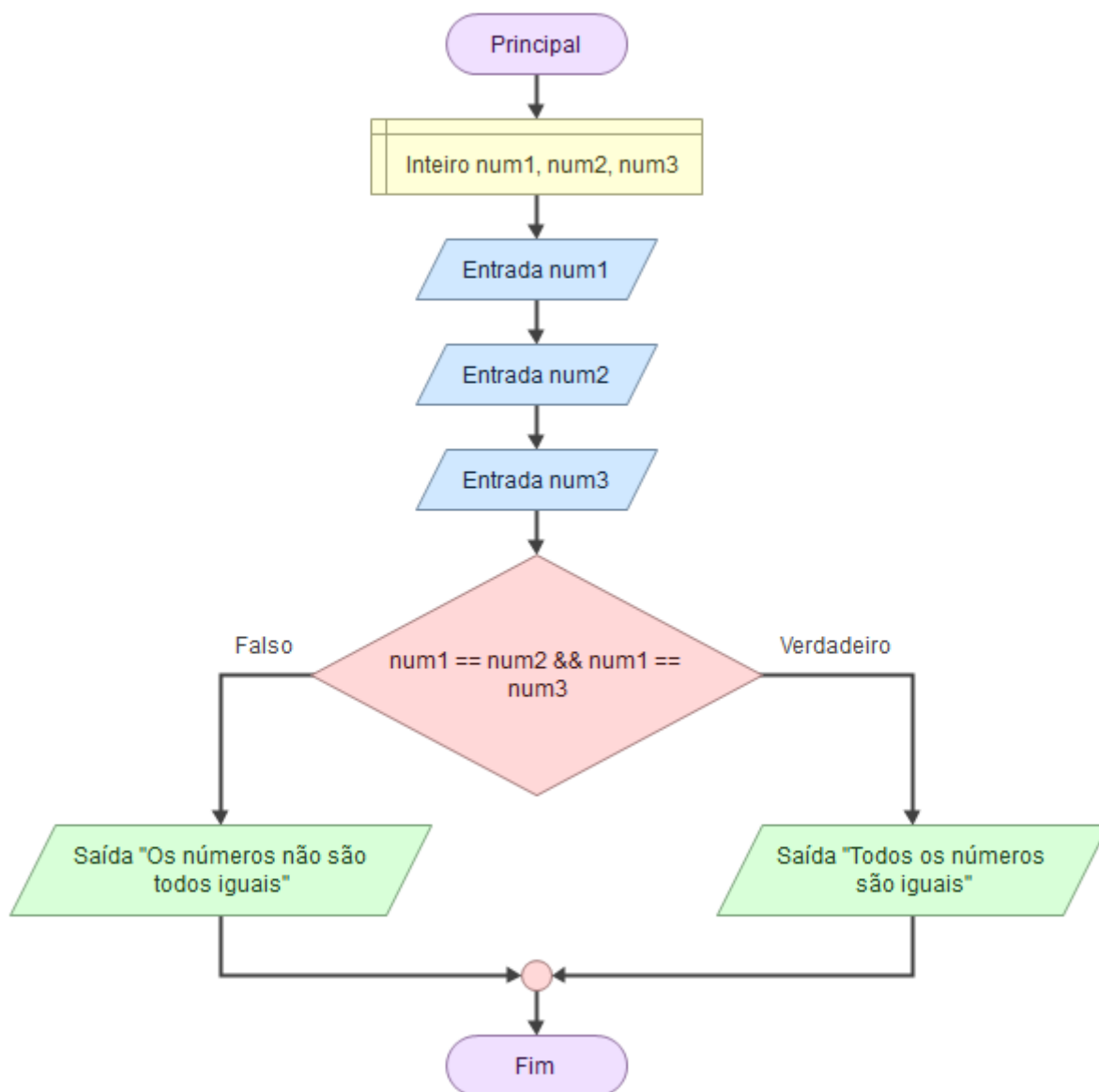
Operadores Matemáticos, Relacionais e Lógicos

2. Faça um algoritmo que leia três números e exiba se todos são iguais.

```
programa
{
    funcao inicio()
    {
        inteiro num1, num2, num3;

        escreva("Digite o primeiro número: ");
        leia(num1);
        escreva("Digite o segundo número: ");
        leia(num2);
        escreva("Digite o terceiro número: ");
        leia(num3);

        se (num1 == num2 && num1 == num3)
        {
            escreva("Todos os números são iguais.\n");
        }
        senao
        {
            escreva("Os números não são todos iguais.\n");
        }
    }
}
```



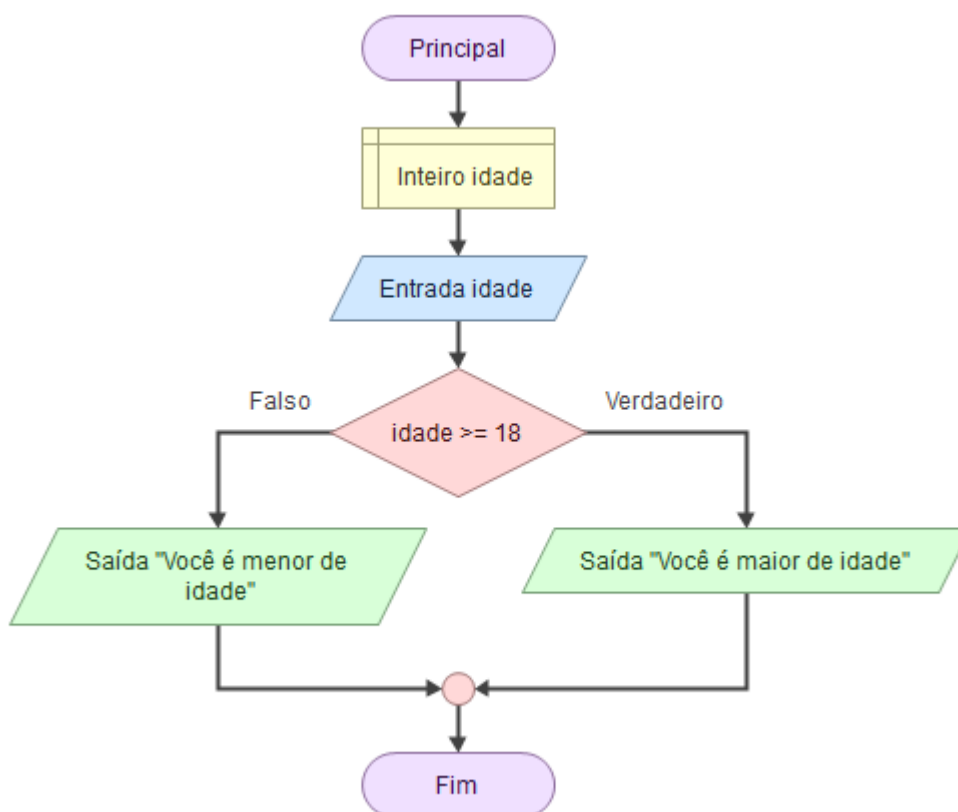
Estruturas de Condição

1. Desenvolva um algoritmo que leia a idade de uma pessoa e determine se ela é maior de idade (idade \geq 18).

```
programa
{
    funcao inicio()
    {
        inteiro idade;

        escreva("Digite sua idade: ");
        leia(idade);

        se (idade  $\geq$  18)
        {
            escreva("Você é maior de idade.\n");
        }
        senao
        {
            escreva("Você é menor de idade.\n");
        }
    }
}
```



Estruturas de Condição

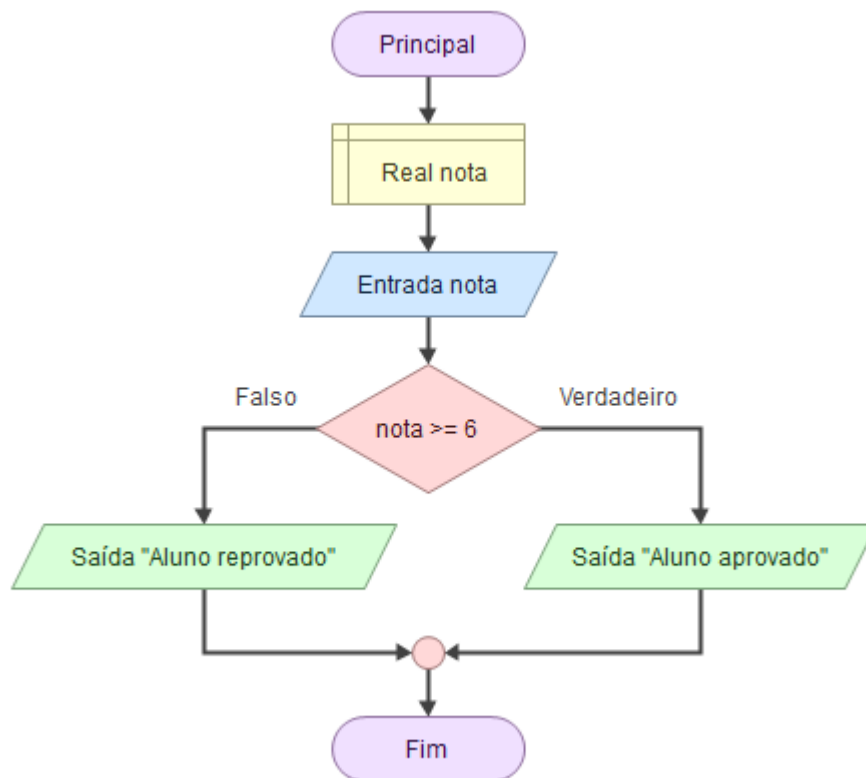
2. Crie um algoritmo que leia a nota de um aluno e exiba se ele foi aprovado (nota ≥ 6) ou reprovado (nota < 6).

```

programa
{
    funcao inicio()
    {
        real nota;

        escreva("Digite a nota do aluno: ");
        leia(nota);

        se (nota >= 6)
        {
            escreva("Aluno aprovado.\n");
        }
        senao
        {
            escreva("Aluno reprovado.\n");
        }
    }
}
  
```



Estruturas de Condição

3. Faça um algoritmo que leia três números e exiba o maior deles.

```

programa
{
    funcao inicio()
    {
        inteiro num1, num2, num3, maior;

        escreva("Digite o primeiro número: ");
        leia(num1);
        escreva("Digite o segundo número: ");
        leia(num2);
        escreva("Digite o terceiro número: ");
        leia(num3);

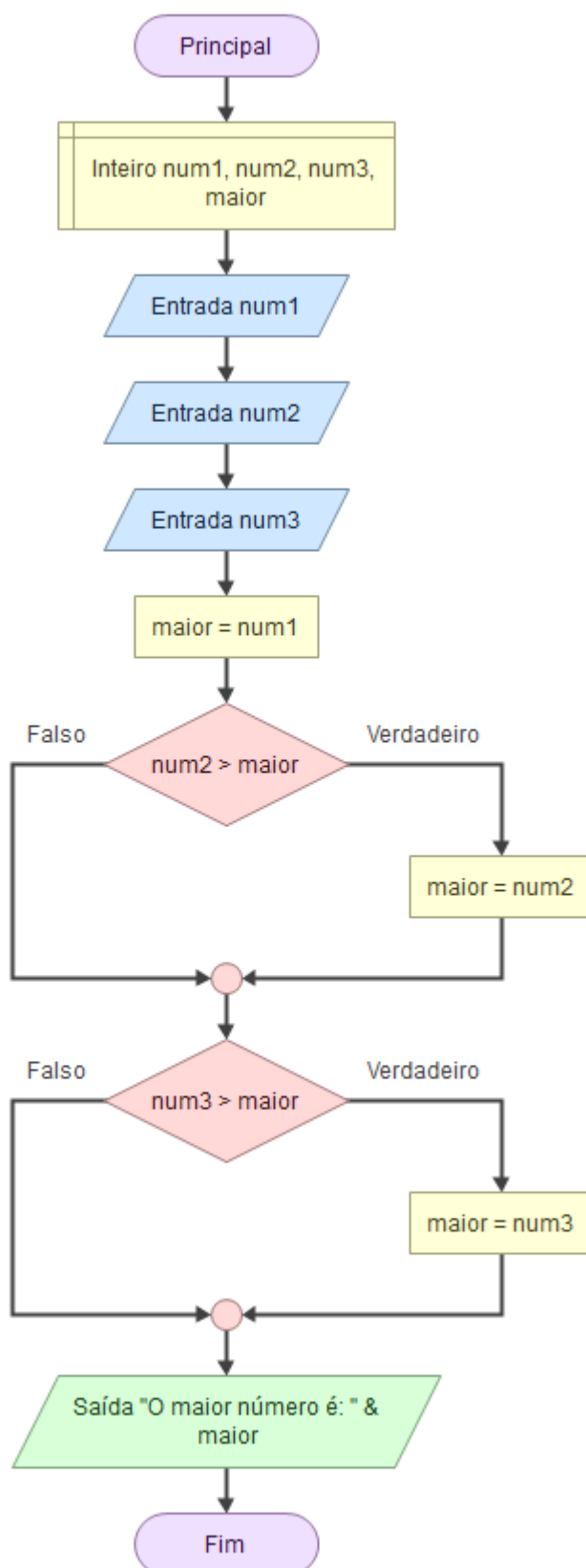
        maior = num1;

        se (num2 > maior)
        {
            maior = num2;
        }
        se (num3 > maior)
        {

```

```
        maior = num3;
    }

    escreva("O maior número é: ", maior, "\n");
}
}
```



Estruturas de Condição

4. Desenvolva um algoritmo que calcule a média de três notas e determine se o aluno está aprovado (média ≥ 6) ou reprovado (média < 6).

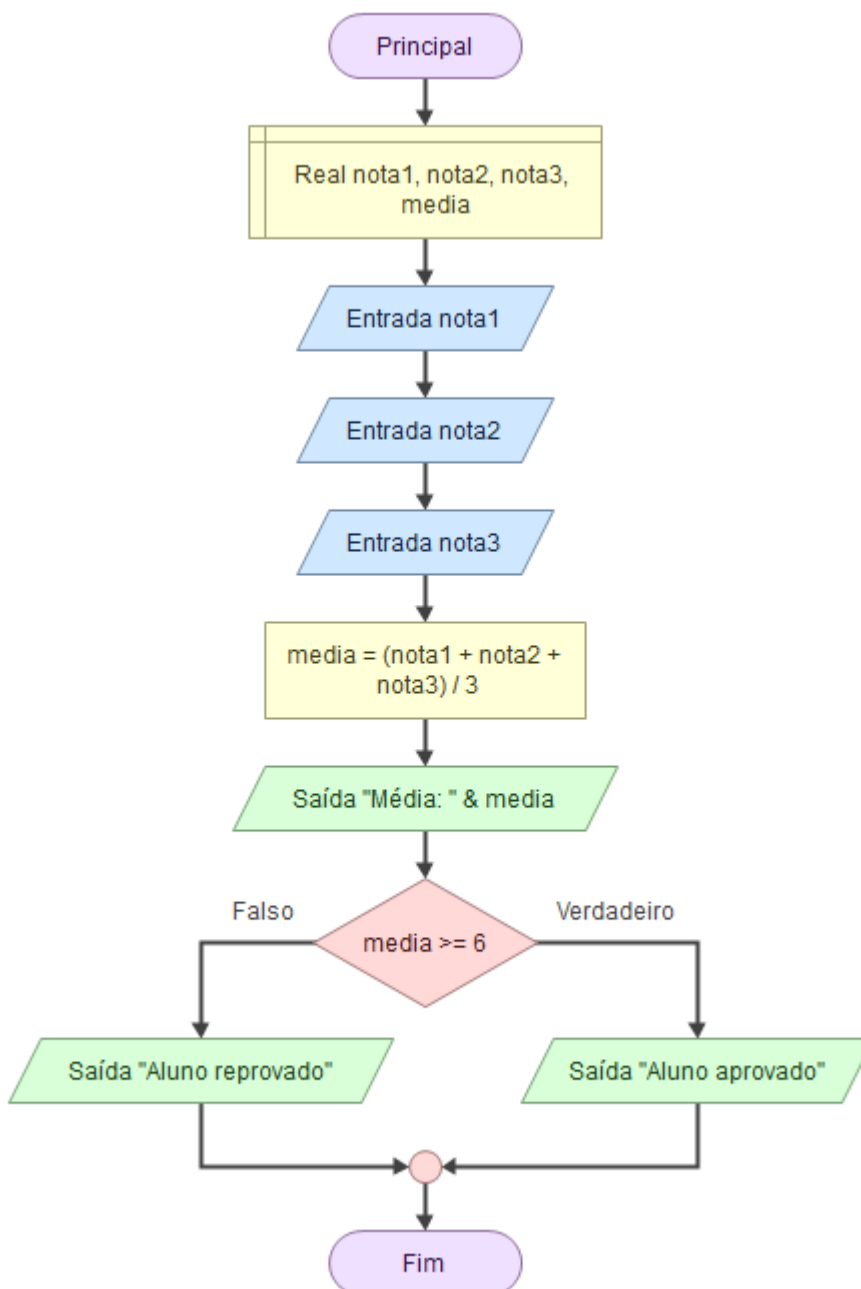
```
programa
{
    funcao inicio()
    {
        real nota1, nota2, nota3, media;

        escreva("Digite a primeira nota: ");
        leia(nota1);
        escreva("Digite a segunda nota: ");
        leia(nota2);
        escreva("Digite a terceira nota: ");
        leia(nota3);

        media = (nota1 + nota2 + nota3) / 3;

        escreva("Média: ", media, "\n");

        se (media  $\geq$  6)
        {
            escreva("Aluno aprovado.\n");
        }
        senao
        {
            escreva("Aluno reprovado.\n");
        }
    }
}
```

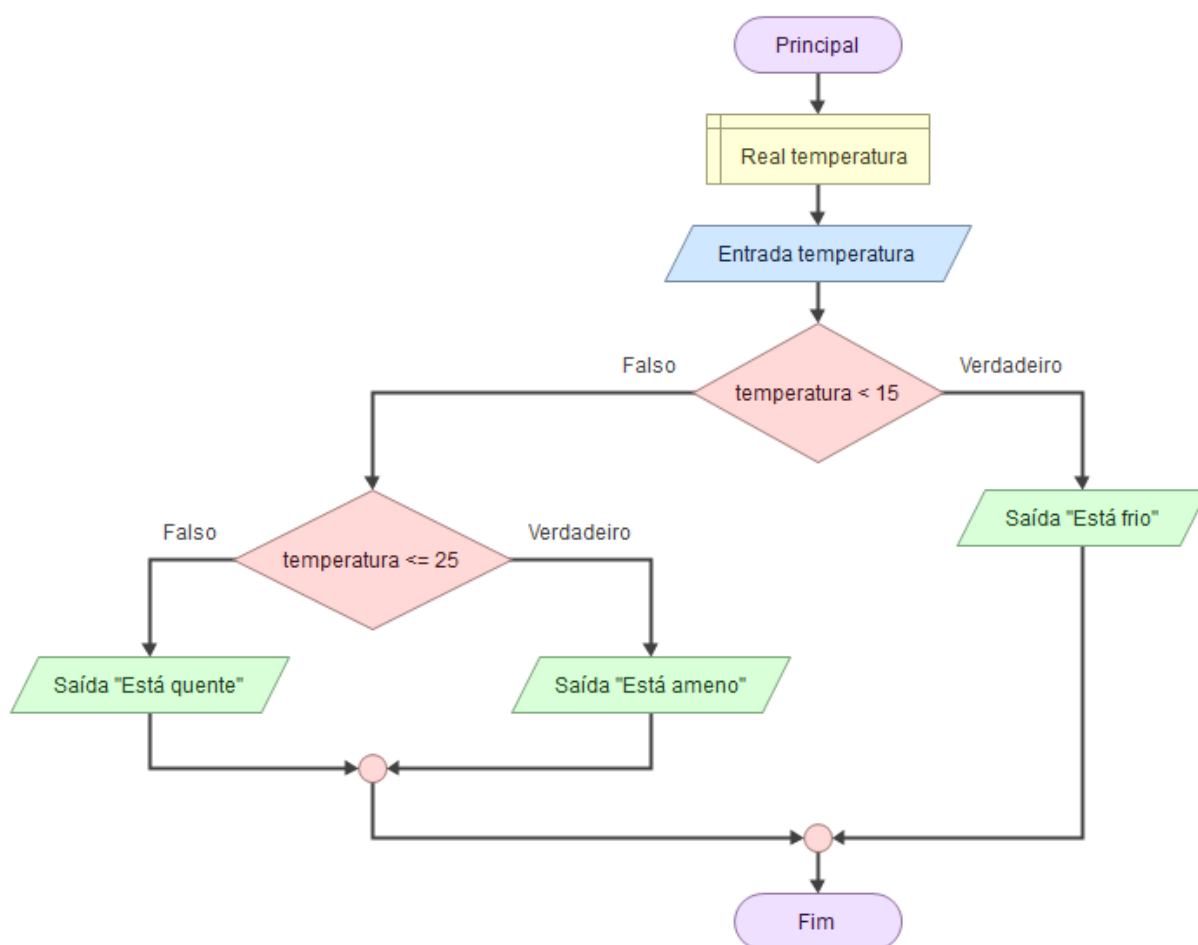
Estruturas de Condição

5. Desenvolva um algoritmo que leia a temperatura em graus Celsius e determine se está frio (abaixo de 15), ameno (entre 15 e 25) ou quente (acima de 25).

```
programa
{
    funcao inicio()
    {
        real temperatura;

        escreva("Digite a temperatura em graus Celsius: ");
        leia(temperatura);

        se (temperatura < 15)
        {
            escreva("Está frio.\n");
        }
        senao se (temperatura >= 15 e temperatura <= 25)
        {
            escreva("Está ameno.\n");
        }
        senao
        {
            escreva("Está quente.\n");
        }
    }
}
```



Estruturas de Repetição

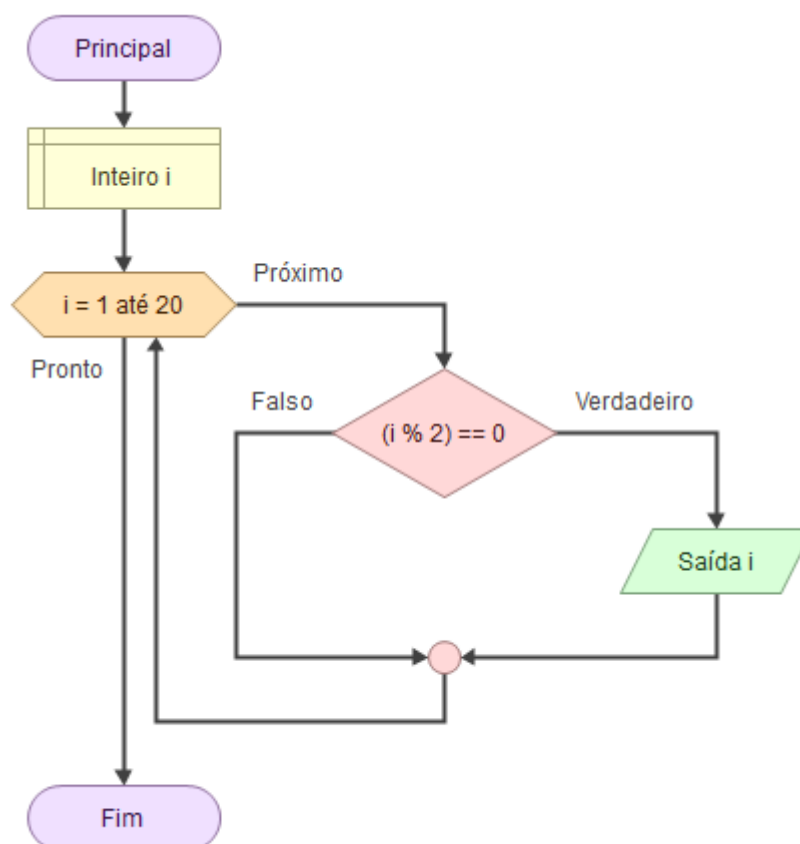
1. Crie um algoritmo que exiba todos os números pares de 1 a 20 usando a estrutura "para".

```

programa
{
    funcao inicio()
    {
        inteiro i;

        para (i = 1; i <= 20; i = i + 1)
        {
            se (i % 2 == 0)
            {
                escreva(i, "\n");
            }
        }
    }
}

```



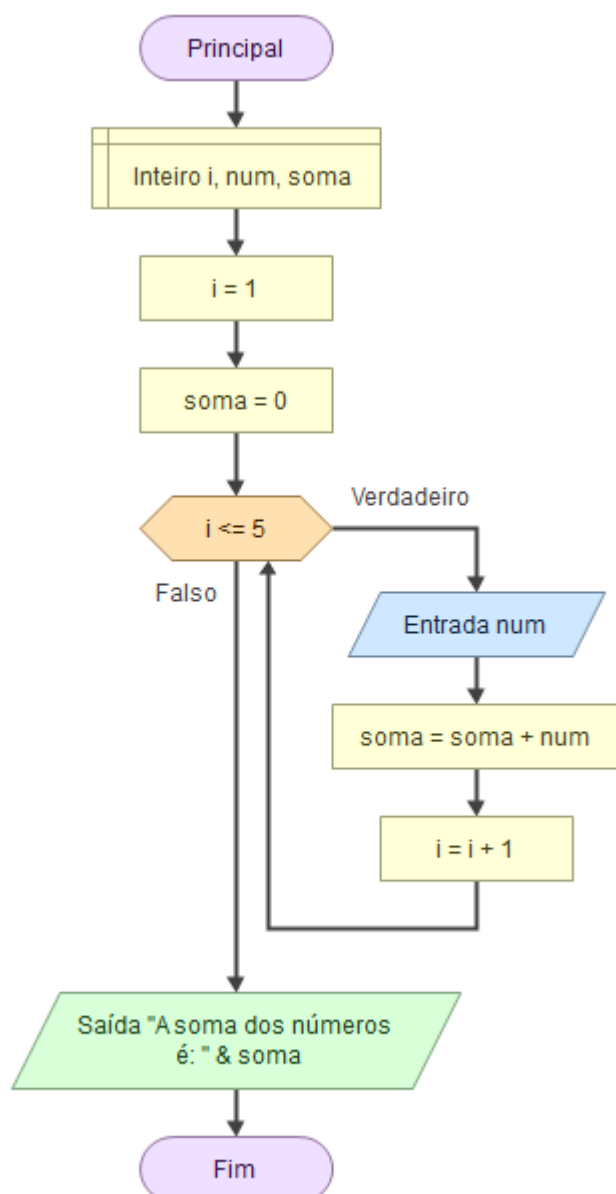
Estruturas de Repetição

2. Faça um algoritmo que leia 5 números e exiba a soma deles usando a estrutura "enquanto".

```
programa
{
    funcao inicio()
    {
        inteiro i = 1, num, soma = 0;

        enquanto (i <= 5)
        {
            escreva("Digite o número ", i, ": ");
            leia(num);
            soma = soma + num;
            i = i + 1;
        }

        escreva("A soma dos números é: ", soma, "\n");
    }
}
```



Estruturas de Repetição

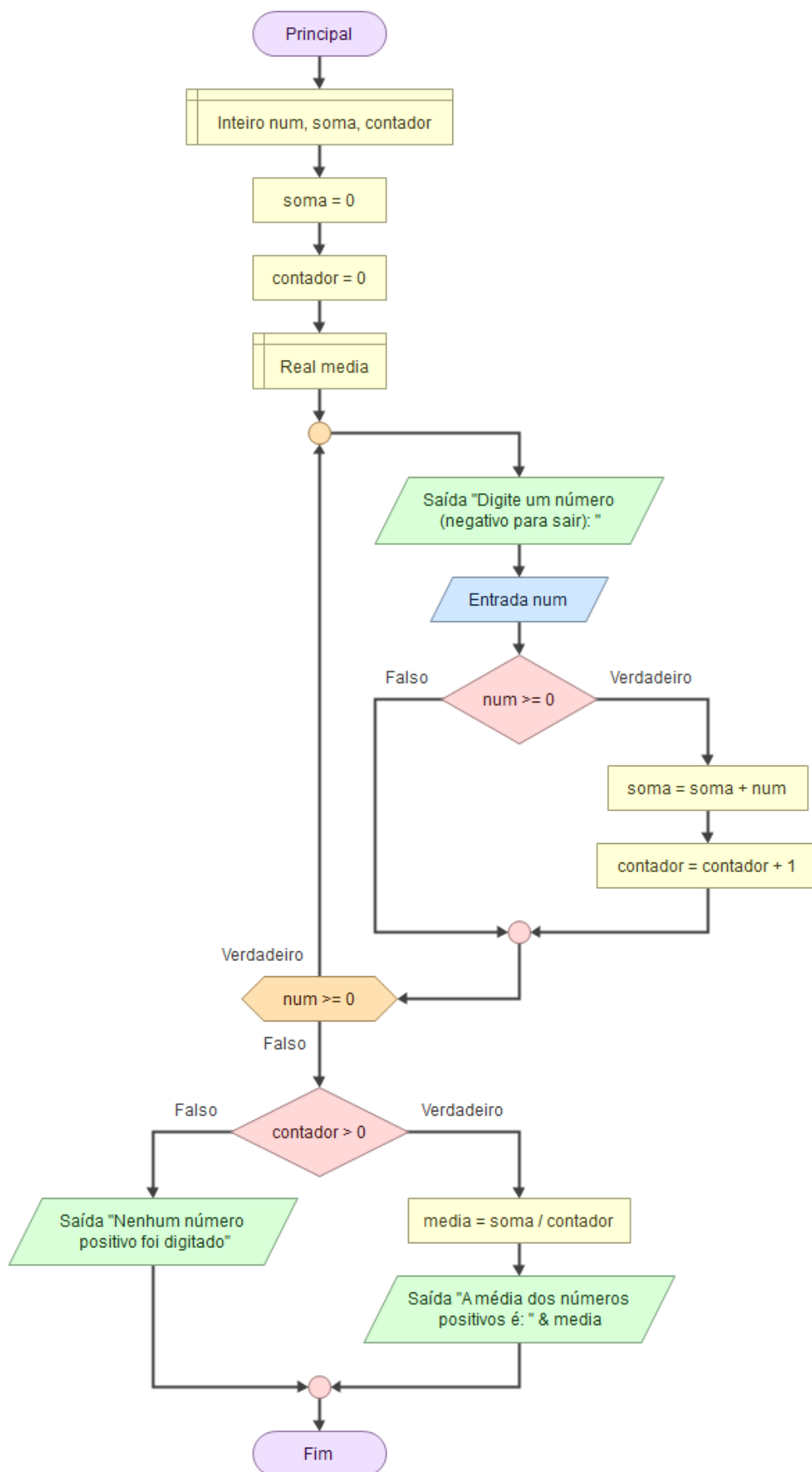
3. Desenvolva um algoritmo que leia números do usuário até que ele digite um número negativo, e então exiba a média dos números positivos digitados usando a estrutura "faça enquanto".

```
programa
{
    funcao inicio()
    {
        inteiro num, soma = 0, contador = 0;
        real media;

        faca
        {
            escreva("Digite um número (negativo para sair): ");
            leia(num);

            se (num >= 0)
            {
                soma = soma + num;
                contador = contador + 1;
            }
        } enquanto (num >= 0)

        se (contador > 0)
        {
            media = soma / contador;
            escreva("A média dos números positivos é: ", media, "\n");
        }
        senao
        {
            escreva("Nenhum número positivo foi digitado.\n");
        }
    }
}
```



Vetores e Matrizes

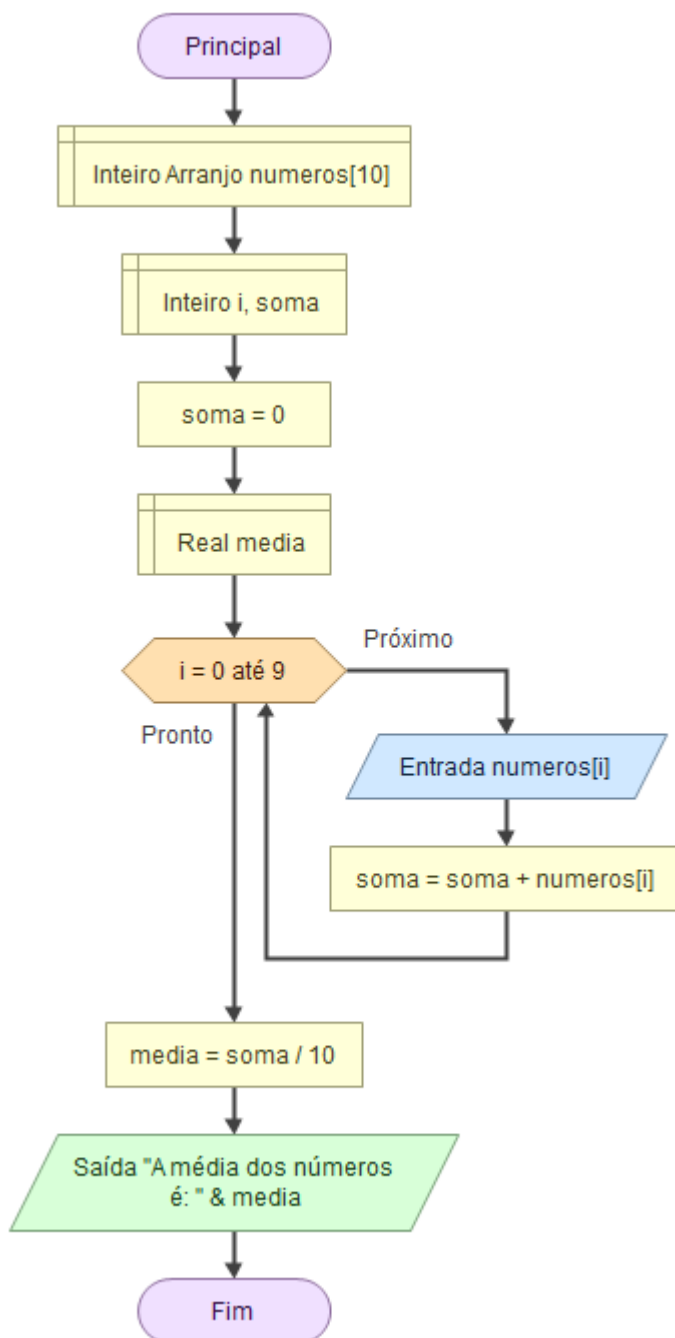
1. Crie um algoritmo que leia 10 números inteiros e exiba a média deles.

```
programa
{
    funcao inicio()
    {
        inteiro numeros[10], i, soma = 0;
        real media;

        para (i = 0; i < 10; i = i + 1)
        {
            escreva("Digite o número ", i + 1, ": ");
            leia(numeros[i]);
            soma = soma + numeros[i];
        }

        media = soma / 10;

        escreva("A média dos números é: ", media, "\n");
    }
}
```



Vetores e Matrizes

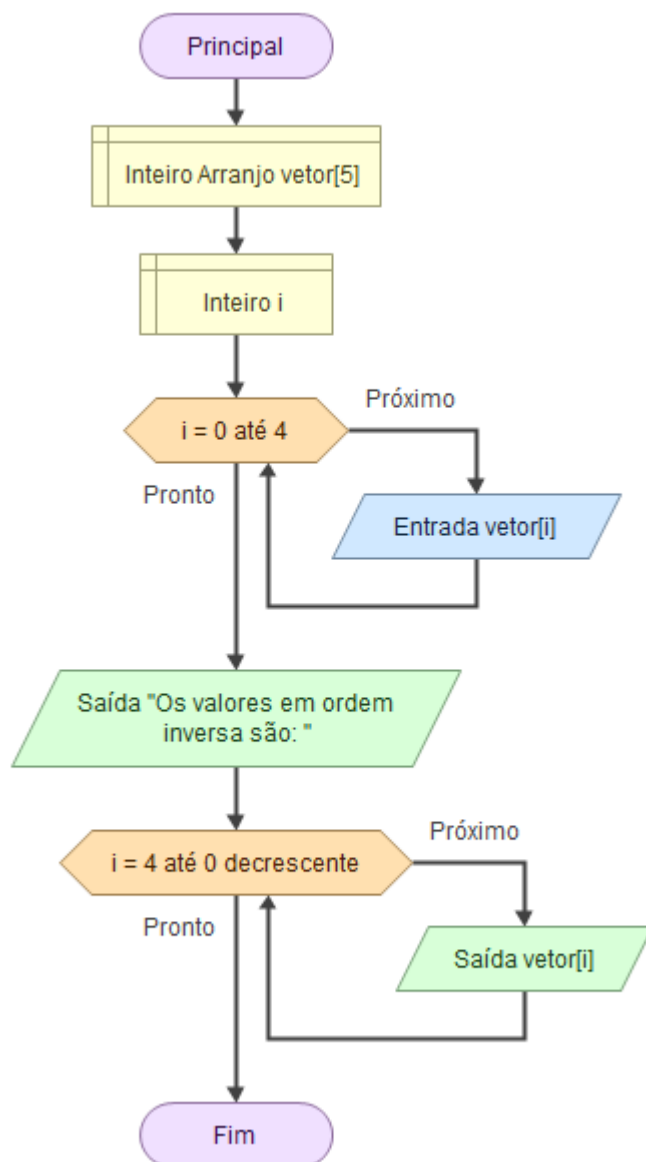
2. Desenvolva um algoritmo que leia um vetor de 5 posições e exiba os elementos em ordem inversa.

```
programa
{
    funcao inicio()
    {
        inteiro vetor[5], i;

        para (i = 0; i < 5; i = i + 1)
        {
            escreva("Digite o valor para a posição ", i + 1, ": ");
            leia(vetor[i]);
        }

        escreva("Os valores em ordem inversa são:\n");

        para (i = 4; i >= 0; i = i - 1)
        {
            escreva(vetor[i], "\n");
        }
    }
}
```



Funções e Procedimentos

1. Crie uma função que receba um número inteiro e retorne seu fatorial.

```
programa
{
    funcao inteiro fatorial(inteiro n)
    {
        inteiro i, resultado = 1;

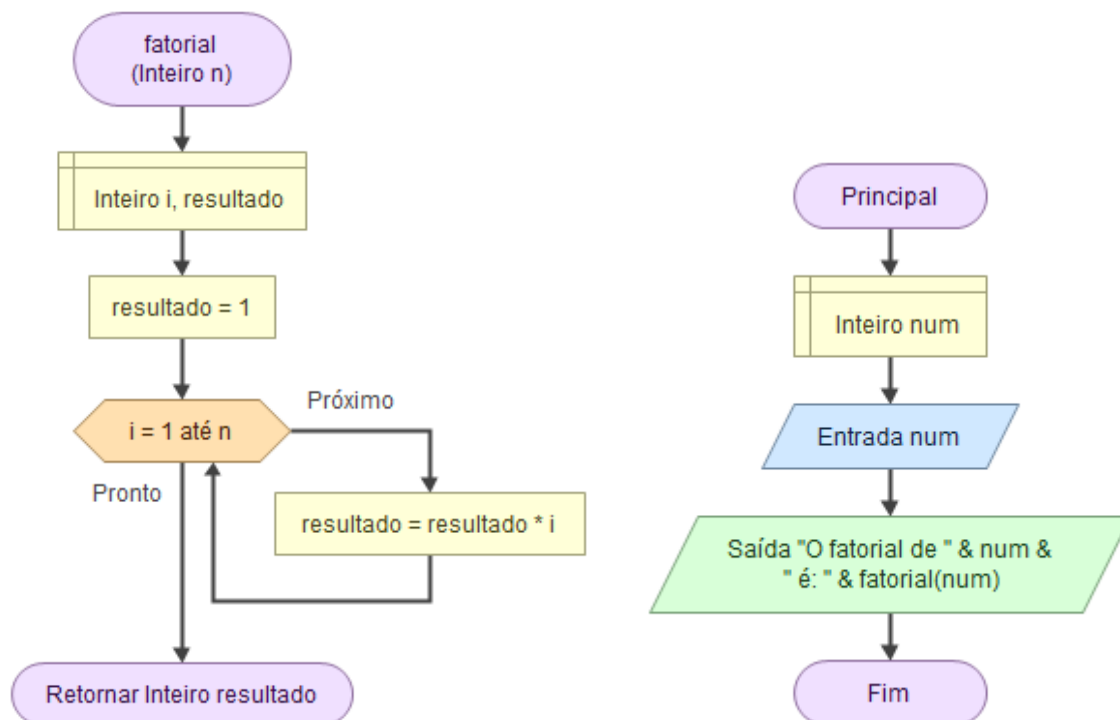
        para (i = 1; i <= n; i = i + 1)
        {
            resultado = resultado * i;
        }

        retorne resultado;
    }

    funcao inicio()
    {
        inteiro num;

        escreva("Digite um número: ");
        leia(num);

        escreva("O fatorial de ", num, " é: ", fatorial(num), "\n");
    }
}
```



Funções e Procedimentos

2. Faça um procedimento que receba um vetor de 10 inteiros e exiba os valores na tela.

```

programa
{
    procedimento exibir_vetor(inteiro v[10])
    {
        inteiro i;

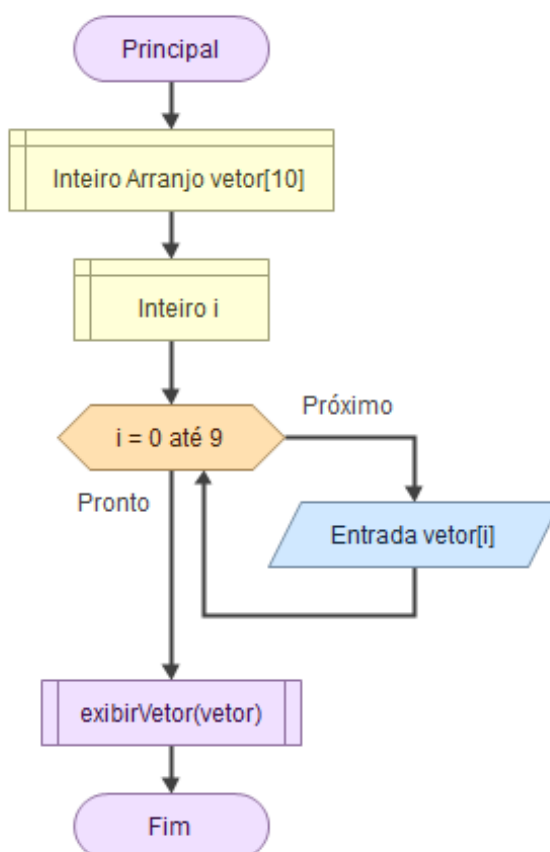
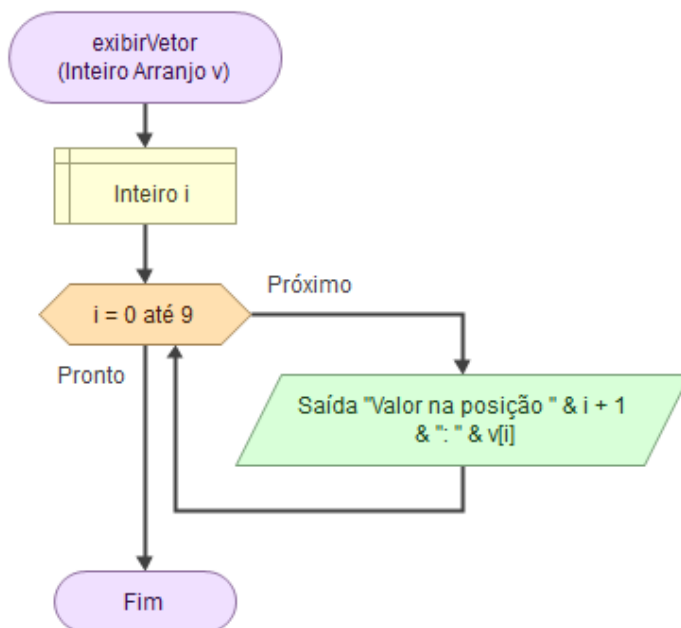
        para (i = 0; i < 10; i = i + 1)
        {
            escreva("Valor na posição ", i + 1, ": ", v[i], "\n");
        }
    }

    funcao inicio()
    {
        inteiro vetor[10], i;

        para (i = 0; i < 10; i = i + 1)
        {
            escreva("Digite o valor para a posição ", i + 1, ": ");
            leia(vetor[i]);
        }
    }
}

```

```
    }  
    exibir_vetor(vetor);  
}  
}
```



Funções e Procedimentos

3. Desenvolva uma função que receba um vetor de 5 reais e retorne a soma dos elementos.

```
programa
{
    funcao real soma_vetor(real v[5])
    {
        inteiro i;
        real soma = 0;

        para (i = 0; i < 5; i = i + 1)
        {
            soma = soma + v[i];
        }

        retorne soma;
    }

    funcao inicio()
    {
        real vetor[5];
        inteiro i;

        para (i = 0; i < 5; i = i + 1)
        {
            escreva("Digite o valor para a posição ", i + 1, ": ");
            leia(vetor[i]);
        }

        escreva("A soma dos valores no vetor é: ", soma_vetor(vetor), "\n");
    }
}
```

