

1. CONCEITOS DE PROGRAMAÇÃO

1.1. LÓGICA

A lógica de programação é necessária para pessoas que desejam trabalhar com implementação de programas para os dispositivos computacionais, ela permite definir a sequência lógica para a elaboração de um projeto.

Então o que é lógica?

Lógica de programação é a técnica de encadear pensamentos para atingir determinado objetivo.

1.2. SEQUÊNCIA LÓGICA

Estes pensamentos podem ser descritos como uma sequência de instruções, que devem ser seguidas para se cumprir uma determinada tarefa.

Sequência Lógica são passos executados até atingir um objetivo ou solução de um problema.

1.3. INSTRUÇÕES

Na linguagem comum, entende-se por instruções **“um conjunto de regras ou normas definidas para a realização ou emprego de algo”**.

Em informática, porém, instrução é a informação que indica a um computador uma ação elementar a executar. Convém ressaltar que uma ordem isolada não permite realizar o processo completo, para isso é necessário um conjunto de instruções colocadas em ordem sequencial lógica.

É evidente que essas instruções têm que ser executadas em uma ordem adequada. Dessa maneira, uma instrução tomada em separado não tem muito sentido; para obtermos o resultado, precisamos colocar em prática o conjunto de todas as instruções, na ordem correta.

Instruções são conjuntos de regras ou normas definidas para a realização ou emprego de algo. Em informática, é o que indica a um computador uma ação elementar a executar.

1.4. ALGORITMO

Um algoritmo é formalmente uma sequência finita de passos que levam a execução de uma tarefa. Podemos pensar em algoritmo como uma receita, uma sequência de instruções que dão cabo de uma meta específica. Estas tarefas não podem ser redundantes nem subjetivas na sua definição, devem ser claras e precisas.

Como exemplos de algoritmos, podemos citar os algoritmos das operações básicas (adição, multiplicação, divisão e subtração) de números reais decimais. Outros exemplos

seriam os manuais de aparelhos eletrônicos, como uma TV, que explicam passo-a-passo como, por exemplo, programar os canais.

Até mesmo as coisas mais simples, podem ser descritas por sequências lógicas. Por exemplo:

“Somar dois números quaisquer”.

- Coloque o primeiro número na caixa A
- Coloque o segundo número na caixa B
- Some o número da caixa A com número da caixa B e coloque o resultado na caixa C



1.5. PROGRAMAS

Os programas de computadores nada mais são do que algoritmos escritos numa linguagem de computador (Java, Pascal, C, COBOL, Fortran, Visual Basic, Smalltalk, entre outras) e que são interpretados e executados por uma máquina, no caso um computador. Notem que dada esta interpretação rigorosa, um programa é por natureza, muito específico e rígido em relação aos algoritmos da vida real.

1.6. EXERCÍCIOS

1) Descreva uma sequência de passos para somar dois números e multiplicar o resultado pelo primeiro número.

2) Faça um algoritmo para trocar uma lâmpada. Descreva com detalhes.

2. DESENVOLVENDO ALGORITMOS

2.1. PSEUDOCÓDIGO

Os algoritmos são descritos em uma linguagem chamada **pseudocódigo** ou **pseudolinguagem**. Este nome é uma alusão à posterior implementação em uma linguagem de programação, ou seja, quando formos programar em uma linguagem, por exemplo C, estaremos gerando código em C. Por isso os algoritmos são independentes das linguagens de programação. Ao contrário de uma linguagem de programação não existe um formalismo rígido de como deve ser escrito o algoritmo.

O algoritmo deve ser fácil de interpretar e fácil de codificar. Ou seja, ele deve ser o intermediário entre a linguagem falada e a linguagem de programação.

2.2. REGRAS PARA CONSTRUÇÃO DO ALGORITMO

Para escrever um algoritmo precisamos descrever a sequência de instruções, de maneira simples e objetiva. Para isso utilizaremos algumas técnicas:

- Usar somente um verbo por frase
- Imaginar que você está desenvolvendo um algoritmo para pessoas que não trabalham com informática
- Usar frases curtas e simples
- Ser objetivo
- Procurar usar palavras que não tenham sentido dúbio

2.3. FASES

É importante ressaltar que qualquer tarefa que siga determinado padrão pode ser descrita por um algoritmo, como por exemplo:

CALCULAR O SALDO FINANCEIRO DE UM ESTOQUE

Entretanto ao montar um algoritmo, precisamos primeiro dividir o problema apresentado em três fases fundamentais.



Onde temos:

ENTRADA: São os dados de entrada do algoritmo

PROCESSAMENTO: São os procedimentos utilizados para chegar ao resultado final

SAÍDA: São os dados já processados

2.4. EXEMPLO DE ALGORITMO

Imagine o seguinte problema: Calcular a média final dos alunos da 3ª Série. Os alunos realizarão quatro provas: P1, P2, P3 e P4.

Onde:

$$\text{Média Final} = (P1+P2+P3+P4)/4$$

Para montar o algoritmo proposto, faremos três perguntas:

a) **Quais são os dados de entrada?**

R: Os dados de entrada são P1, P2, P3 e P4.

b) **Qual será o processamento a ser utilizado?**

R: O procedimento será somar todos os dados de entrada e dividi-los por 4 (quatro)

$$(P1+P2+P3+P4)/4$$

c) **Quais serão os dados de saída?**

R: O dado de saída será a média final

Algoritmo

- Receba a nota da prova1
- Receba a nota de prova2
- Receba a nota de prova3
- Receba a nota da prova4
- Some todas as notas e divida o resultado por 4
- Mostre o resultado da divisão

2.5. EXERCÍCIOS

1) Identifique os dados de entrada, processamento e saída no algoritmo abaixo:

- Receba código da peça
- Receba valor da peça
- Receba Quantidade de peças
- Calcule o valor total da peça (Quantidade * Valor da peça)
- Mostre o código da peça e seu valor total

2) Faça um algoritmo para “Calcular o estoque médio de uma peça”, sendo que:

$$\text{ESTOQUEMÉDIO} = (\text{QUANTIDADE MÍNIMA} + \text{QUANTIDADE MÁXIMA}) / 2$$

3. DIAGRAMA DE BLOCO

3.1. O QUE É UM DIAGRAMA DE BLOCO?

O diagrama de blocos é uma forma padronizada e eficaz para representar os passos lógicos de um determinado processamento. Com o diagrama podemos definir uma sequência de símbolos, com significado bem definido, portanto, sua principal função é facilitar a visualização dos passos de um processamento.





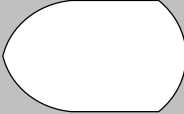
Para desenvolver um diagrama correto de forma clara e precisa, algumas recomendações são necessárias:

- Inicie o diagrama com poucos blocos, e ao poucos vá subdividindo os blocos de forma a criar pequenos trechos.
- Recomenda-se iniciar a construção de cima para baixo e da direita para a esquerda
- Não cruze as linhas entre os blocos
- Transcreva o diagrama de bloco em pseudolinguagem

3.2. SIMBOLOGIA

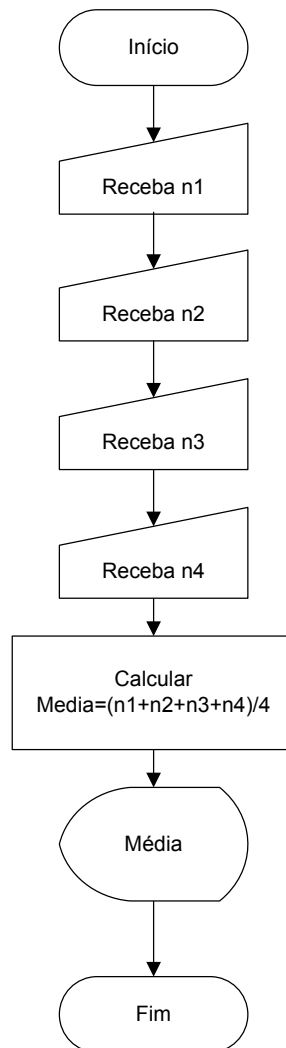
De forma a tornar mais simples o entendimento de diagramas de blocos, sé necessária uma padronização dos blocos que serão utilizados, alguns os blocos utilizados são demonstrados na Tabela 1.

Tabela 1 – Principais Blocos

	Terminal – símbolo utilizado como ponto para indicar o início e/ou o fim de um programa
	Seta de fluxo de dados – permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos existentes
	Processamento – Símbolo ou bloco que se utiliza para indicar cálculos (algoritmos) a efetuar atribuições de valores ou qualquer manipulação de dados que tenha um bloco específico para sua descrição
	Teclado – Entrada de dados que serão inseridos através do teclado
	Display – Saída de dados visuais através de um monitor.

Dentro do símbolo sempre terá algo escrito, pois somente os símbolos não nos dizem nada. Veja nos exemplos a seguir:

“CALCULAR A MÉDIA DE 4 NOTAS”



3.3. EXERCÍCIOS

1) Construa um diagrama de blocos que:

- Leia a cotação do dólar
- Leia um valor em dólares
- Converta esse valor para Real
- Mostre o resultado

2) Desenvolva um diagrama que:

- Leia 4 (quatro) números
- Calcule o quadrado para cada um
- Some todos e
- Mostre o resultado

3) Construa um diagrama de blocos para pagamento de comissão de vendedores de peças, levando-se em consideração que sua comissão será de 5% do total da venda e que você tem os seguintes dados:

- Identificação do vendedor
- Código da peça
- Preço unitário da peça
- Quantidade vendida

4. CONSTANTES, VARIÁVEIS E TIPOS DE DADOS.

Variáveis e constantes são os elementos básicos que um programa manipula. Uma variável é um espaço reservado na memória do computador para armazenar um tipo de dado determinado.

4.1. CONSTANTES

Constante é um determinado valor fixo que não se modifica ao longo do tempo, durante a execução de um programa.

4.2. VARIÁVEIS

Variável é a representação simbólica dos elementos de certo conjunto. Cada variável corresponde a uma posição de memória, cujo conteúdo pode se alterado ao longo do tempo durante a execução de um programa. Embora uma variável possa assumir diferentes valores, ela só pode armazenar um valor a cada instante.

Partindo de uma analogia, uma variável pode ser considerada como uma caixa, onde o tipo de dado que pode ser armazenado na variável é o tamanho da caixa, mesmo princípio acontece em programação, uma variável que irá receber um valor lógico tem um tamanho em bytes (1 byte) inferior a uma variável que irá receber um valor Real (4 ou 8 bytes).

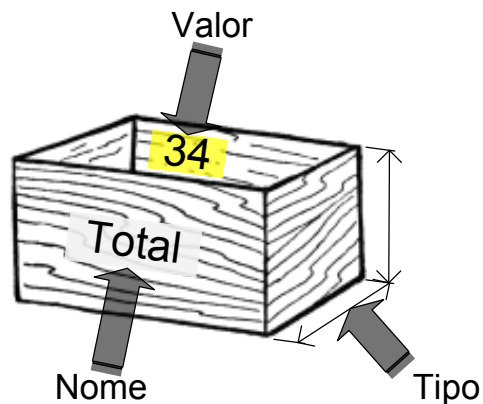


Figura 1 - Representação de uma variável

Uma variável é composta por três partes:

1. **Nome:** é o rótulo da caixa; como ela irá ser identificada.
2. **Valor:** o que está armazenado dentro da caixa.
3. **Tipo:** são as dimensões da caixa, em computação se refere a quantidade de bits disponíveis para armazenamento.

4.3. NOME DAS VARIÁVEIS

Para atribuir o nome para uma variável, algumas regras devem se obedecer:

1. O nome de uma variável pode ter um ou mais caracteres;
2. O primeiro caractere do nome deverá obrigatoriamente ser uma letra ou sublinhado;
3. O nome não poderá conter espaços em branco;
4. Não poderão ser utilizados outros caracteres diferentes de letras, números ou sublinhado;
5. Não poderão ser utilizadas como nome da variável, palavras reservadas da linguagem, a Tabela 2 apresenta a lista de palavras reservadas no C.

Tabela 2 - Palavras-chave do C

<i>auto</i>	<i>double</i>	<i>int</i>	<i>struct</i>
<i>break</i>	<i>else</i>	<i>long</i>	<i>switch</i>
<i>case</i>	<i>enum</i>	<i>register</i>	<i>typedef</i>
<i>char</i>	<i>extern</i>	<i>return</i>	<i>union</i>
<i>const</i>	<i>float</i>	<i>short</i>	<i>unsigned</i>
<i>continue</i>	<i>for</i>	<i>signed</i>	<i>void</i>
<i>default</i>	<i>goto</i>	<i>sizeof</i>	<i>volatile</i>
<i>do</i>	<i>if</i>	<i>static</i>	<i>while</i>

4.4. TIPOS DE VARIÁVEIS

As variáveis e as constantes precisam ter seu tipo especificado para armazenar e recuperar os valores, para isso os seguintes tipos são disponíveis em C:

Tabela 3 - Tipos de variáveis

Tipo	Descrição
char	Caractere. Campo de 8 bits que são usados para representação dos caracteres
double	Número em ponto flutuante (números reais) de precisão dupla (aproximadamente 14 dígitos significativos) os valores vão de $\pm 1.7 \times 10^{-308}$ até $\pm 1.7 \times 10^{308}$
float	Número em ponto flutuante de precisão simples (aproximadamente 7 dígitos significativos) os valores vão de $\pm 3,4 \times 10^{-38}$ até $\pm 3,4 \times 10^{38}$
int	Inteiro de 32 bits com sinal (de -2.147.483.648 até 2.147.483.647)
void	Sem valor

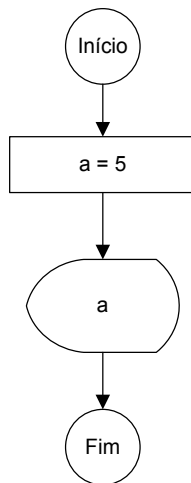
4.5. LENDO E IMPRIMINDO VALORES EM C

Para ler e imprimir valores em C não tem muito segredo, o principal detalhe que deve se observar se refere ao tipo de variável, pois as funções de captura e impressão necessitam que seja especificado esse tipo.

Vamos a um exemplo:

4.5.1. Imprimindo valores

A impressão de valores é feita pelo comando *printf*:



```
int main()
{
    int a = 5;
    printf("%d \n", a);
    return 0;
}
```

Figura 2 - Imprimindo valores

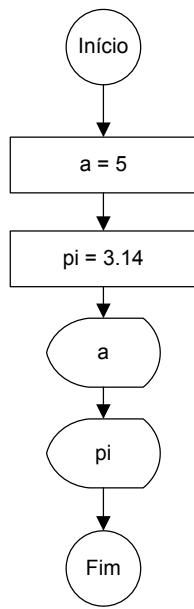
Perceba que o *printf* possui alguns caracteres estranhos dentro das aspas, o primeiro %d se refere ao tipo de dado esperado, outros formatos que poderão ser utilizados:

Tabela 4 - Caracteres coringas

Caractere	Descrição
%d ou %i	Número decimal inteiro.
%f	Número real
%c	Caractere

O “\n” (barra invertida) é o símbolo indicado para ao final realizar uma quebra de linha na hora de apresentar o resultado

Outro exemplo de impressão:



```

int main()
{
    int a = 5;
    float pi = 3.14;
    printf("A = %d PI = %f\n", a, pi);
    return 0;
}
  
```

Figura 3 - Imprimindo 2 valores

Numa operação desses valores, fizemos duas saídas *printfs*, uma com o resultado sendo impresso como inteiro e outra como número real:

```

int main()
{
    int a = 5;
    float pi = 3.14;
    printf("Resultado 1= %d\n", a * pi);
    printf("Resultado 2= %f\n", a * pi);
    return 0;
}
  
```

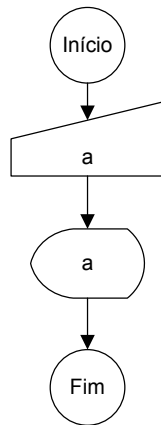
Você irá perceber que o Resultado 1 será inconsistente, isso se deve a forma de armazenamento do valor Real que na hora de realizar a apresentação não é apresentado devidamente:

```

Resultado 1= -2147483648
Resultado 2= 15.700001
  
```

4.5.2. Lendo um valor

Para ler um valor digitado, utilizamos a função *scanf*:



```
int main()  
{  
    int a;  
    scanf("%d", &a);  
    printf("Valor de a: %d\n", a);  
    return 0;  
}
```

Figura 4 - Lendo um valor

Note as semelhanças dos argumentos com a função *printf*, a exceção se dá no uso do operador “&” que em C é utilizado para passar a localização de uma variável na memória e não seu valor.

Cuide que ao executar esse programa a tela apresentada está vazia, aguardando que você digite algo:

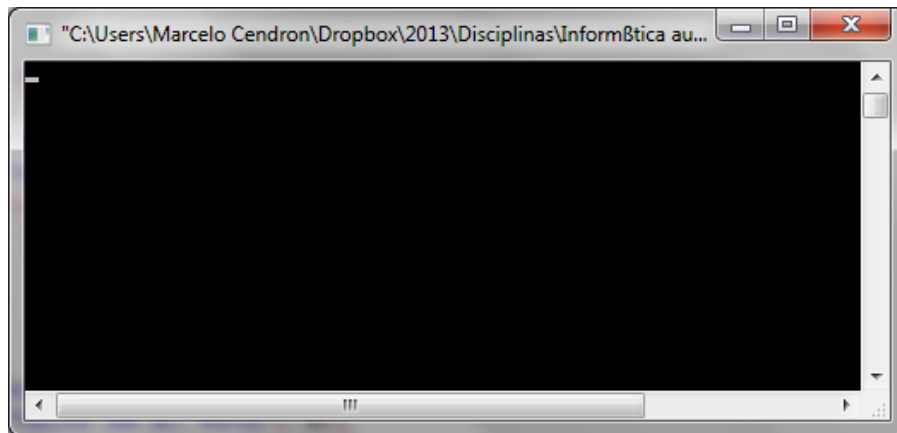


Figura 5 - Aguardando algo ser digitado

Após digitar o valor:

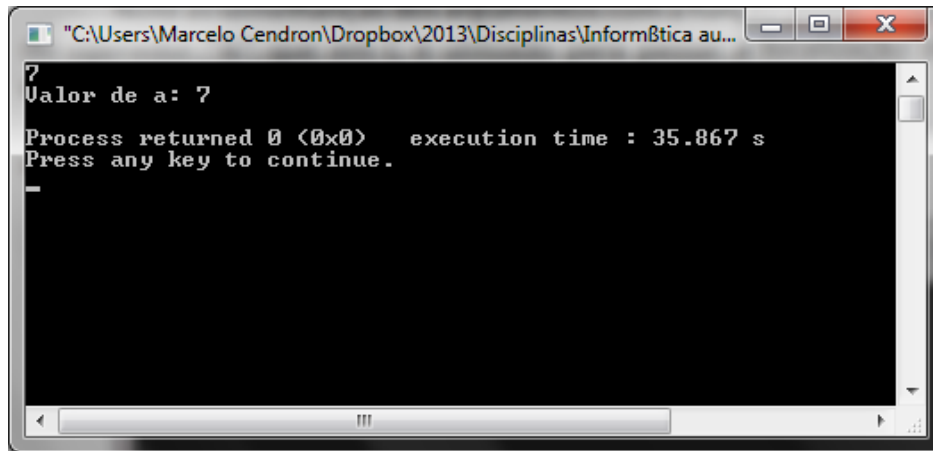
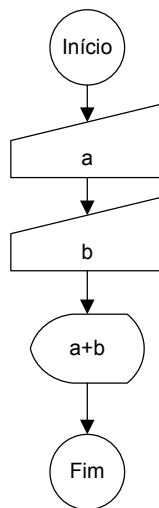


Figura 6 - Após a execução

4.6. EXEMPLO 01 – LENDO DOIS VALORES E APRESENTANDO A SOMA

Nesse exemplo iremos apresentar um programa que irá solicitar ao usuário que digite dois valores e apresente a soma deles:



```

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

    printf("Resultado = %d\n", a+b);
    return 0;
}
  
```

Figura 7 - Algoritmo para somar dois valores

4.7. EXERCÍCIOS

1. Converta o exemplo acima de forma que possa realizar a soma de números reais.
2. Indique os nomes das variáveis que são válidos. Justifique os nomes inválidos.

a. peso	d. int	g. area.do.quadrado
b. média_final	e. 1dia	h. valor real
c. R\$	f. teste 1	i. a+b

5. OPERADORES

Os operadores são meios pelo qual incrementamos, decrementamos, comparamos e avaliamos dados dentro do computador. Temos três tipos de operadores:

- Operadores Aritméticos
- Operadores Relacionais
- Operadores Lógicos

5.1. OPERADORES ARITMÉTICOS

Os operadores aritméticos são os utilizados para obter resultados numéricos. Além da adição, subtração, multiplicação e divisão, há o operador resto da divisão. Os símbolos para os operadores aritméticos são:

Tabela 5 - Operadores aritméticos

Operação	Símbolo
Adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto da divisão	%

Tabela 6 - Hierarquia das Operações Aritméticas

Nível	Operação
1º	() Parênteses
2º	* ou / (o que aparecer primeiro)
3º	+ ou - (o que aparecer primeiro)

Exemplo

Total = PRECO * QUANTIDADE

$1 + 7 * 2 - 1 = 14$

$3 * (1 - 2) + 4 * 2 = 5$

5.2. OPERADORES RELACIONAIS

Os operadores relacionais são utilizados para comparar números. Os valores a serem comparados podem ser números ou variáveis.

Estes operadores sempre retornam valores lógicos (0 para falso e 1 para verdadeiro). Para estabelecer prioridades no que diz respeito a qual operação executar primeiro, utilize os parênteses.

Os operadores relacionais são:

Tabela 7 - Operadores relacionais

Descrição	Símbolo
Igual a	= ou ==
Diferente	!= ou <>
Maior que	>
Menor que	<
Maior ou igual a	>=
Menor ou igual a	<=

Exemplo:

Tendo duas variáveis A = 5 e B = 3

Os resultados das expressões seriam:

Tabela 8 - Exemplo de expressões

Expressão	Resultado
A == B	FALSO
A != B	VERDADEIRO
A > B	VERDADEIRO
A < B	FALSO
A >= B	VERDADEIRO
A <= B	FALSO

5.3. OPERADORES LÓGICOS

Os operadores lógicos servem para combinar resultados de expressões, retornando se o resultado final é verdadeiro ou falso.

Os operadores lógicos são:

Tabela 9 - Operadores lógicos

Operador	Original	Em C
e	and	&&
ou	or	(tecla ao lado do Z no teclado)
não	not	!

E/AND = Uma expressão AND (E) é verdadeira se todas as condições forem verdadeiras

Tabela 10 - Operador E

1º Valor	2º Valor	Resultado
F	F	F

F	V	F
V	F	F
V	V	V

OR/OU = Uma expressão OR (OU) é verdadeira se pelo menos uma condição for verdadeira

Tabela 11 - Operador OU

1º Valor	2º Valor	Resultado
F	F	F
F	V	V
V	F	V
V	V	V

NOT = Uma expressão NOT (NÃO) inverte o valor da expressão ou condição, se verdadeira inverte para falsa e vice-versa.

Tabela 12 - Operador NÃO

1º Valor	Resultado
V	F
F	V

Exemplos:

Suponha que temos três variáveis A = 5, B = 8 e C = 1.

Os resultados das expressões seriam:

Tabela 13 - Expressões

Expressão			Resultado
A == B	&&	B > C	FALSO
A != B	 	B < C	VERDADEIRO
A > B	!		VERDADEIRO
A < B	&&	B > C	VERDADEIRO
A >= B	 	B == C	FALSO
A <= B	!		FALSO

5.4. EXERCÍCIOS

1) Tendo as variáveis SALARIO, IR e SALLIQ, e considerando os valores abaixo. Informe se as expressões são verdadeiras ou falsas.

SALARIO	IR	SALLIQ	EXPRESSÃO	V ou F
100,00	0,00	100	(SALLIQ >= 100,00)	
200,00	10,00	190,00	(SALLIQ < 190,00)	
300,00	15,00	285,00	SALLIQ = SALARIO - IR	

2) Sabendo que A=3, B=7 e C=4, informe se as expressões abaixo são verdadeiras ou falsas.

- a) $(A + C) > B$ ()
- b) $B \geq (A + 2)$ ()
- c) $C == (B - A)$ ()
- d) $(B + A) \leq C$ ()
- e) $(C + A) > B$ ()

3) Sabendo que A=5, B=4 e C=3 e D=6, informe se as expressões abaixo são verdadeiras ou falsas.

- a) $(A > C) \ \&\& \ (C \leq D)$ ()
- b) $(A+B) > 10 \ \|\ (A+B) == (C+D)$ ()
- c) $(A \geq C) \ \&\& \ (D \geq C)$ ()

4) Faça um programa que solicite ao usuário que digite a base e a altura de um triângulo. Em seguida, apresente a área do mesmo.

$$\text{Área} = (\text{Base} * \text{Altura}) / 2$$

5) Crie um programa que solicite ao usuário que digite a quantidade de horas e de minutos. Após isso, o programa deverá responder a quantidade de segundos nesse período.

$$\text{Segundos} = (\text{Quantidade de horas} * 3600) + (\text{Quantidade de minutos} * 60)$$

6) Escrever um algoritmo para determinar o consumo médio de um automóvel sendo fornecida a distância total percorrida pelo automóvel e o total de combustível gasto

$$\text{Consumo médio} = \text{distância total} / \text{total de combustível gasto}$$

7) A Loja Café com Açúcar está vendendo seus produtos em 5 (cinco) prestações sem juros. Faça um programa que receba um valor de uma compra e mostre o valor das prestações.

6. COMANDOS DE DECISÃO

Os comandos de decisão ou desvio fazem parte das técnicas de programação que conduzem a estruturas de programas que não são totalmente sequenciais. Com as instruções de SALTO ou DESVIO pode-se fazer com que o programa proceda de uma ou outra maneira, de acordo com as decisões lógicas tomadas em função dos dados ou resultados anteriores.

6.1. SE / IF

A estrutura de decisão “SE/IF” normalmente vem acompanhada de um comando, ou seja, se determinada condição for satisfeita pelo comando SE/IF então execute determinado comando.



A decisão é sempre composta por um ou mais operador relacional, e quando mais de um operador relacional estiver na condição, obrigatoriamente deve-se utilizar o operador lógico para conectar as expressões.

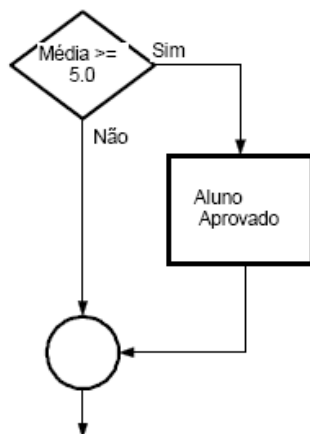


A decisão só pode ser respondida com Sim ou Não

Imagine um algoritmo que determinado aluno somente estará aprovado se sua média for maior ou igual a 5.0, veja no exemplo de algoritmo como ficaria.

**SE MEDIA >= 5.0 ENTÃO
ALUNO APROVADO**

Em diagrama de blocos ficaria assim:



```
int main()
{
    int media = 7;
    if(media >= 5)
    {
        printf("Aprovado");
    }
    return 0;
}
```

Figura 8 - Comando IF

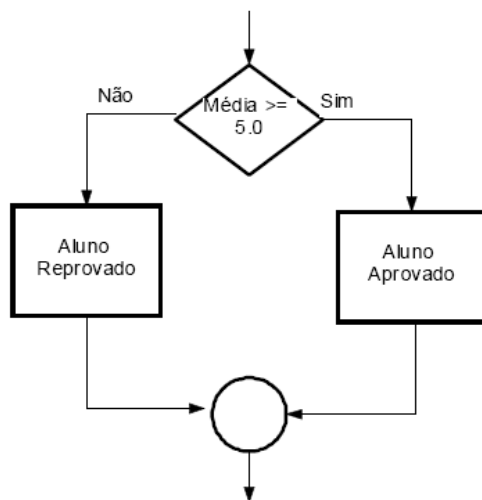
6.2. SE... SENÃO / IF....ELSE

A estrutura de decisão “SE /SENÃO”, funciona exatamente como a estrutura “SE”, com apenas uma diferença, em “SE” somente podemos executar comandos caso a condição seja verdadeira, diferente de “SE/SENÃO”, pois sempre um comando será executado independente da condição, ou seja, caso a condição seja “verdadeira” o comando da condição será executado, caso contrário o comando da condição “falsa” será executada.

Em algoritmo ficaria assim:

SE MÉDIA >= 5.0 ENTÃO
ALUNO APROVADO
SENÃO
ALUNO REPROVADO

O diagrama e o código em C



```
int main()
{
    int media = 7;
    if(media >= 5)
    {
        printf("Aprovado");
    }
    else
    {
        printf("Reprovado");
    }
    return 0;
}
```

Figura 9 - Uso do SE/SENÃO

No exemplo acima está sendo executada uma condição que, se for verdadeira, executa o comando “APROVADO”, caso contrário executa o segundo comando “REPROVADO”.

6.2.1. Exercício

1) Faça um programa que verifique se um caractere é M ou F. Se for M deve aparecer a mensagem “Masculino”, se for F, deve aparecer “Feminino”.

2) Faça um programa que some dois valores, se o resultado for 0 deve aparecer a mensagem “Zero”, caso contrário, “Diferente de Zero”.

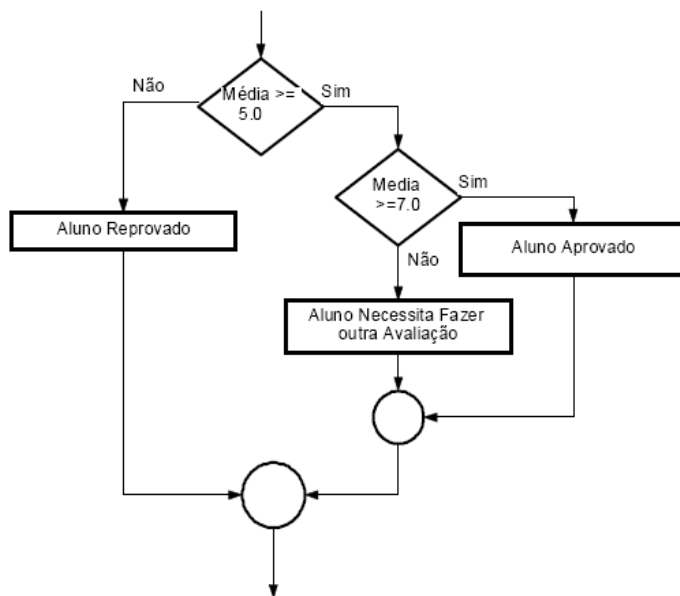
6.3. IF/ELSE ENCADEADOS

Para o exemplo dado anteriormente podemos pensar em tornar o programa mais elaborado, criando as seguintes situações:

- **Caso o aluno tire nota abaixo de 5, seja considerado reprovado.**
- **Caso o aluno tenha nota maior ou igual a 5, porém, menor do que 7 estará em exame.**
- **Caso o aluno tenha nota maior ou igual a 7 estará aprovado.**

Atente a saída do primeiro IF, nele a condição **média >= 5** garante duas situações, se falsa então apresenta a mensagem de “Reprovado”, se for verdadeira entra para mais uma decisão IF.

No segundo IF não me preocupo se a média for maior do que 5, pois isso foi verificado no bloco anterior, com isso verifico agora se **média >= 7** e as duas possibilidades: “Exame” ou “Aprovado”.

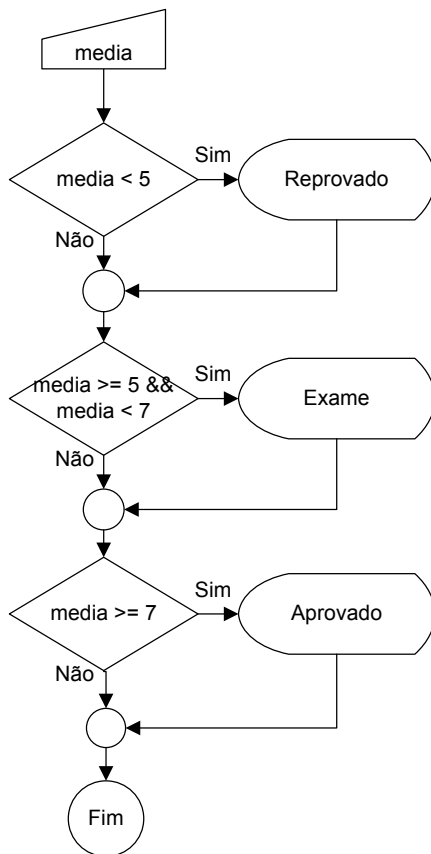


```
int main()
{
    int media = 6;
    if(media >= 5)
    {
        if(media >= 7)
        {
            printf("Aprovado");
        }
        else
        {
            printf("Exame");
        }
    }
    else
    {
        printf("Reprovado");
    }
    return 0;
}
```

Figura 10 - IFs encadeados

6.4. VÁRIOS IFs

Podemos também utilizar vários IFs em sequência para resolver o problema da média:



```

int main()
{
    int media;
    printf("Digite a media: ");
    scanf("%d", &media);
    if(media < 5)
    {
        printf("Reprovado");
    }
    if(media >= 5 && media < 7)
    {
        printf("Exame");
    }
    if(media >= 7)
    {
        printf("Aprovado");
    }
    return 0;
}

```

Figura 11 - IF em sequência

Nesse caso, deve-se tomar cuidado nas condições do IF, como queremos que apenas uma mensagem seja exibida, a condição não pode ser aplicada para vários casos, por isso, que no segundo IF tivemos que realizar duas comparações.

6.5. CASO SELECIONE / SELECT... CASE

A estrutura de decisão CASO/SELECIONE é utilizada para testar, na condição, uma única expressão, que produz um resultado, ou, então, o valor de uma variável, em que está armazenado um determinado conteúdo. Compara-se, então, o resultado obtido no teste com os valores fornecidos em cada cláusula “Caso”.

No exemplo do diagrama de blocos abaixo, é recebido uma variável “**Valor**” e testado seu conteúdo, caso uma das condições seja satisfeita, é apresentado o mês correspondente, caso contrário é apresentada a mensagem “Não é um mês válido!”.

Diagrama de Bloco:

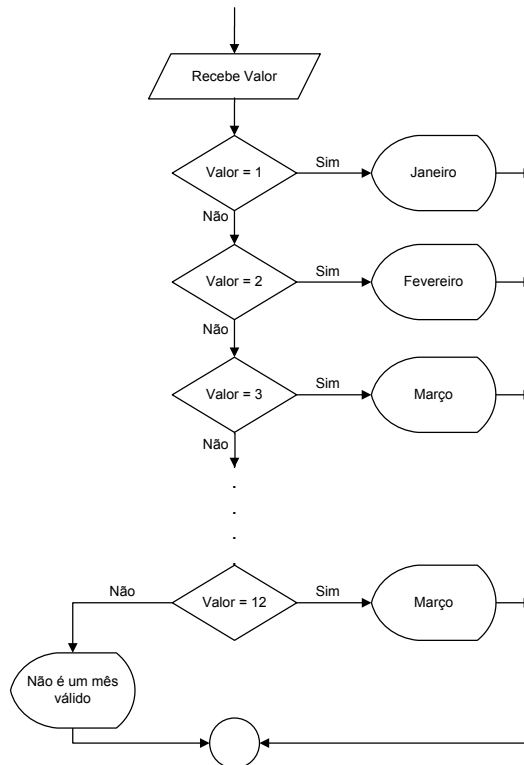


Figura 12 - Diagrama de bloco do Switch

Em C:

```

int main()
{
    int valor = 6;

    switch (valor) {
        case 1: printf("Janeiro"); break;
        case 2: printf("Fevereiro"); break;
        case 3: printf("Março"); break;
        case 4: printf("Abril"); break;
        case 5: printf("Maio"); break;
        case 6: printf("Junho"); break;
        case 7: printf("Julho"); break;
        case 8: printf("Agosto"); break;
        case 9: printf("Setembro"); break;
        case 10: printf("Outubro"); break;
        case 11: printf("Novembro"); break;
        case 12: printf("Dezembro"); break;

        default: printf("Não é um mês válido!"); break;
    }
    return 0;
}

```

Figura 13 - Código fonte do switch

6.6. EXERCÍCIOS

1. Os moradores de uma localidade possuem um poço artesiano que distribui água para a comunidade, a cobrança é feita pelo consumo, até 10m^3 o morador paga taxa fixa de R\$10,00, acima dessa quantidade, ele paga R\$2,00 por metro cúbico excedente. Faça um programa que receba a quantidade de litros gastos pelo morador e retorne o valor que deve ser pago.

2. Faça um diagrama de bloco que leia um número inteiro e mostre uma mensagem indicando se este número é par ou ímpar, e se é positivo ou negativo.

3. A Secretaria de Meio Ambiente que controla o índice de poluição mantém 3 grupos de indústrias que são altamente poluentes do meio ambiente. O índice de poluição aceitável varia de 0,05 até 0,3. Se o índice sobe para mais de 0,3 as indústrias do 1º grupo são intimadas a suspenderem suas atividades, se o índice crescer para mais de 0,4 as indústrias do 1º e 2º grupo são intimadas a suspenderem suas atividades, se o índice atingir mais de 0,5 todos os grupos deverão ser notificados a paralisarem suas atividades. Faça um programa que leia o índice de poluição medido e emita a notificação adequada aos diferentes grupos de empresas.

4. Elabore um algoritmo que dada a idade de um ciclista classifique-o em uma das seguintes categorias:

Infantil A = 5 a 7 anos

Infantil B = 8 a 11 anos

Juvenil A = 12 a 13 anos

Juvenil B = 14 a 17 anos

Adultos = Maiores de 18 anos

7. COMANDOS DE REPETIÇÃO

Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado prevalecer ou até que seja alcançado.



Esse tipo de estrutura é muito importante em programação, e seu uso é constante nas disciplinas de programação.

Basicamente um laço de repetição é composto por três partes:

1. Inicialização
2. Verificação da condição.
3. Incremento/decremento

No diagrama a seguir são apresentadas as três partes:

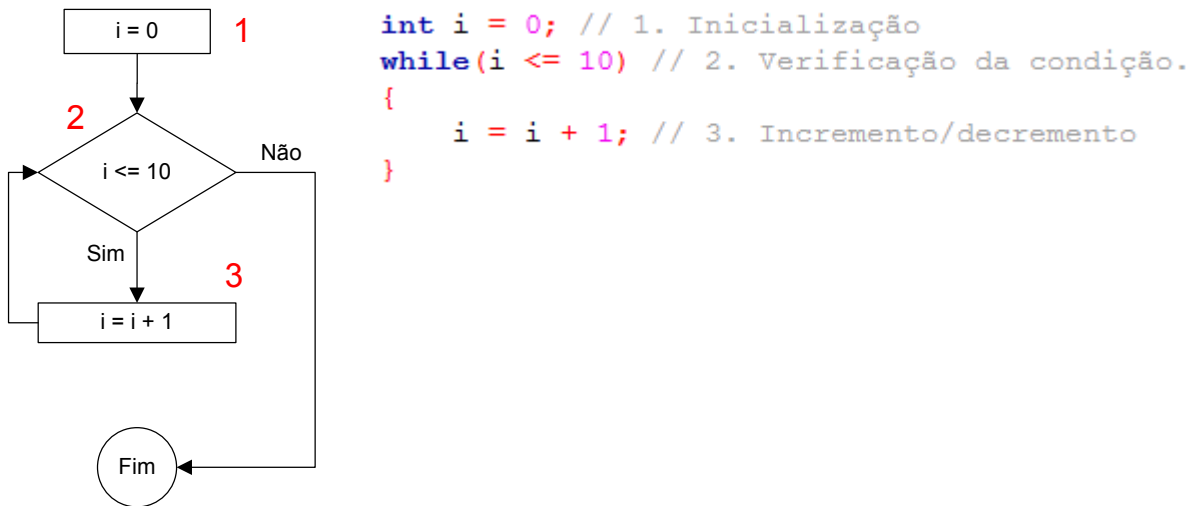


Figura 14 - Partes de um laço de repetição

Quanto ao formato da repetição, são basicamente três os formatos aceitos em C:

- Enquanto x, processar (**While... Loop**);
- Processar..., Enquanto x (**Do... While**);
- Para... Até... Seguinte (**For... To... Next**).

7.1. ENQUANTO X, PROCESSAR (WHILE... LOOP).

Neste caso, o bloco de operações será executado enquanto a condição x for verdadeira. O teste da condição será sempre realizado antes de qualquer operação. Enquanto a condição for verdadeira o processo se repete. Podemos utilizar essa estrutura para trabalharmos com contadores.

Em diagrama de bloco a estrutura é a seguinte:

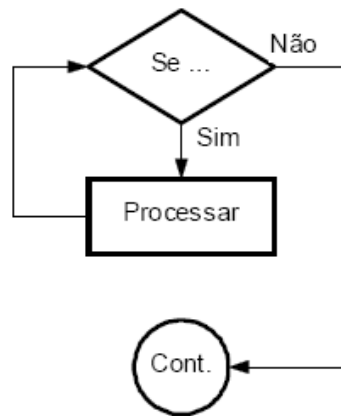
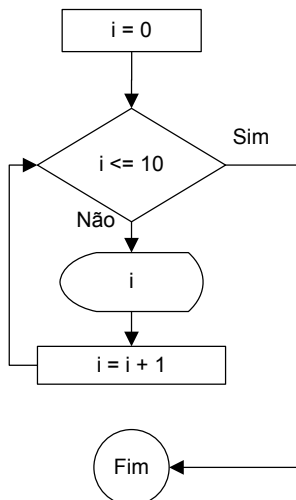


Figura 15 - Diagrama While

Exemplo de contador:



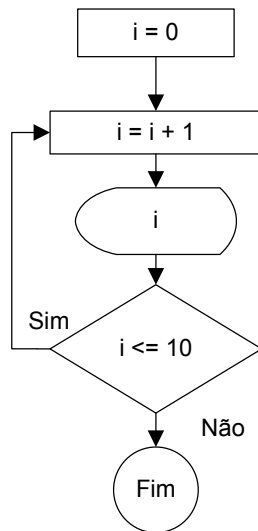
```
int main()
{
    int i = 0;
    while(i <= 10)
    {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
```

Figura 16 - Laço While

7.2. PROCESSAR..., ENQUANTO X (DO... LOOP WHILE).

Neste caso primeiro são executados os comandos, e somente depois é realizado o teste da condição. Se a condição for verdadeira, os comandos são executados novamente, caso seja falso é encerrado o comando DO.

Em diagrama de bloco e linguagem C:



```
int main()
{
    int i = 0;
    do
    {
        i = i + 1;
        printf("%d\n", i);
    }while(i <= 10);
    return 0;
}
```

Figura 17 - Laço de repetição DO

7.3. PARA... ATÉ... SEGUINTE (FOR... TO... NEXT).

A construção do Laço **For**, em diagrama é muito semelhante com o laço **While**, mas a inicialização, controle e incremento se dá na mesma linha, tornando mais fácil a visualizar.

Abaixo segue o exemplo do laço **While**, modificado para trabalhar com **For**:

Em C:

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



Em certas linguagens há operadores específicos para o incremento como é o caso de C e as linguagens derivadas que utilizam o operador ++ (por exemplo, poderíamos utilizar *i++* em vez de *i = i + 1*).

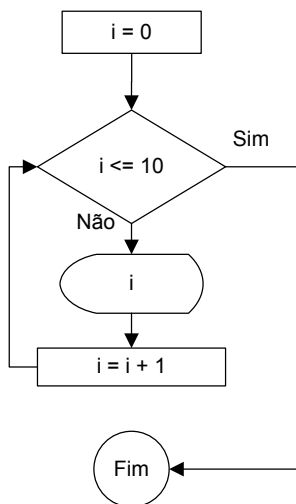
7.4. CASOS PRÁTICOS COM REPETIÇÃO

Para melhor ilustrar o uso dos laços de repetição iremos apresentar alguns exemplos práticos:

7.4.1. Exemplo 01 - Somando os valores

Atividade: Criar um programa para somar os **n** números pares.

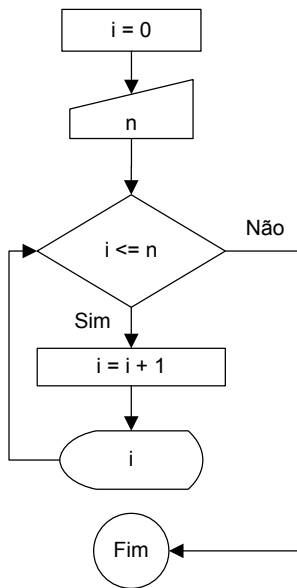
Partimos do laço de repetição já apresentando *While*:



```
int main()
{
    int i = 0;
    while(i <= 10)
    {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
```

Figura 18 - Laço While

Como não sabemos o valor de **n**, iremos criar uma variável para ela, especificar um valor e utilizar o valor de n como limite do laço de repetição.



```

int main()
{
    int i = 0;
    int n;
    printf("Digite o valor de N: ");
    scanf("%d", &n);
    while(i <= n)
    {
        i = i + 1;
        printf("%d\n", i);
    }
    return 0;
}

```

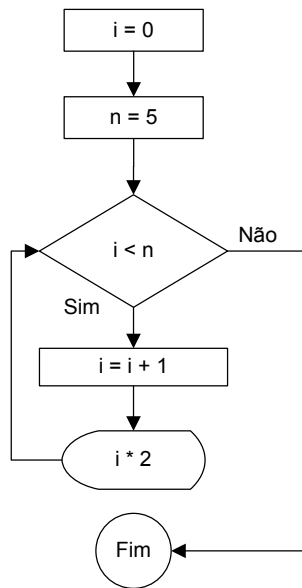
Figura 19 - Modificações no laço básico

Se for digitado o valor 5 (utilizaremos esse valor para o restante do exemplo) perceba que o programa irá contar de 1 a 6, não coincidindo com o esperado que era contar até 5, para resolver isso, vamos retirar o sinal de “=” da comparação.

A atividade proposta é somar os números pares, desconsiderando o 0 (zero) o programa deve realizar a seguinte conta:

$$\begin{aligned} \text{Total} &= 2 + 4 + 6 + 8 + 10 \\ \text{Total} &= 30 \end{aligned}$$

Antes de realizar a soma, vamos apresentar esses valores, como nosso programa apresenta os números de 1 a 5, para apresentar os valores pares basta multiplicarmos o valor de *i* por 2:



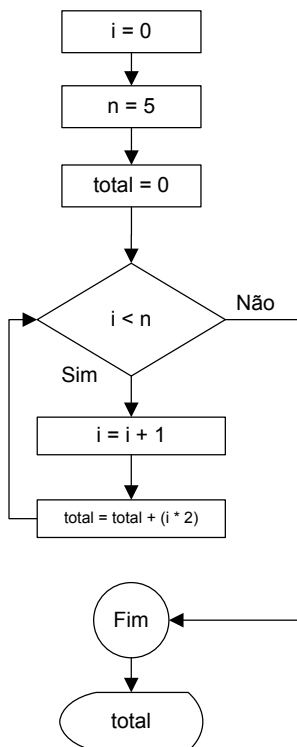
```

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

Figura 20 - Imprimindo valores pares

Por enquanto apenas alteramos o valor mostrado.

Agora vamos criar uma variável temporária que irá armazenar para cada passo do laço o dobro do valor de *i* e vamos deslocar o printf para o final:



```

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

Executando passo-a-passo a repetição se desdobra na seguinte forma (note que após o cálculo, o resultado é armazenado em **total**):

Etapa	Variável Total	Variável i	Cálculo efetuado
1º	0	1	$0 + (1 * 2) = 2$
2º	2	2	$2 + (2 * 2) = 6$
3º	6	3	$6 + (3 * 2) = 12$
4º	12	4	$12 + (4 * 2) = 20$
5º	20	5	$20 + (5 * 2) = 30$
Saída	30		

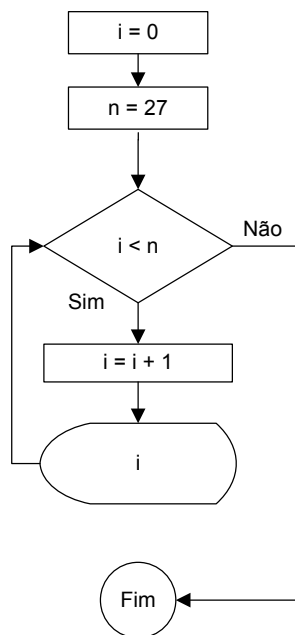
Matematicamente falando a fórmula do nosso exemplo é:

$$Total = \sum_{i=1}^5 i * 2$$

7.4.2. Exemplo 02 – Verificando os divisores.

Atividade: Dado um número **n**, verificar entre **2** e **n – 1** quais são seus divisores.

Para facilitar a resolução vamos partir utilizando o laço de repetição *While*:

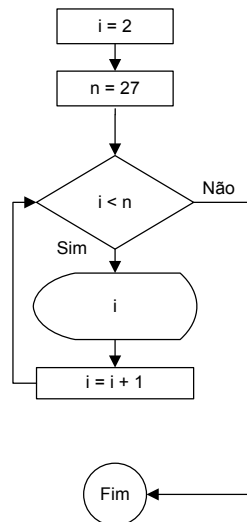


```

int main()
{
    int i = 0;
    int n = 27;
    while(i < n)
    {
        i = i + 1;
        printf("%d\n", i);
    }
    return 0;
}
  
```

Figura 21 - Laço While

Como não precisamos os número 1 e o n, vamos fazer uma modificação na ordem do conteúdo do While, começamos por imprimir o número n (com isso, apresentará o valor de $n - 1$) e iniciamos o valor de $i = 2$:



```

int main()
{
    int i = 2;
    int n = 27;
    while(i < n)
    {
        printf("%d\n", i);
        i = i + 1;
    }
    return 0;
}
  
```

Figura 22 - Modificações no laço While

Agora, iremos verificar se o valor de n pode ser dividido por i e a saída, caso positivo:

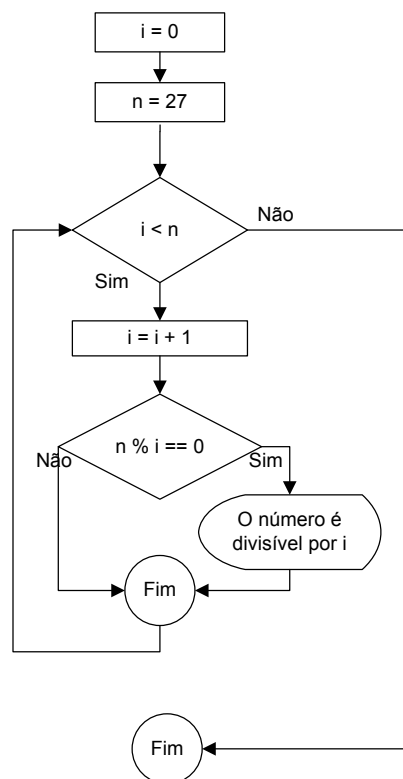


Figura 23 - Diagrama do programa

```

int main()
{
    int i = 2;
    int n = 27;
    while(i < n)
    {
        if(n%i == 0)
        {
            printf("O número %d é divisível por %d\n", n, i);
        }
        i = i + 1;
    }
    return 0;
}

```

Figura 24 - Código-fonte

7.5. EXERCÍCIOS

- 1) Apresente o total da soma obtido dos cem primeiros números inteiros (1+2...+99+100).
- 2) Faça um programa que a partir de dois números imprima os números do intervalo excluindo os valores dados.
- 3) Crie um programa que dado um número ele imprima a tabuada desse número de 1 a 10.
- 4) Faça um programa que conte de 1 a 100 e a cada múltiplo de 10 emita uma mensagem: "Múltiplo de 10".
- 5) Escreva um método que recebe dois números reais **a** e **b** e retorna a soma de todos os números pares existentes entre esses dois
- 6) Utilizando repetição, calcule o fatorial de um número. Lembre-se que o fatorial de 0 é 1 e, não existe fatorial de números negativos.