INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA CATARINENSE – CÂMPUS LUZERNA.

APOSTILA DE PROGRAMAÇÃO EM OCTAVE

SUMÁRIO

1. 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.

1.1. CONSTANTES

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

1.2. 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).

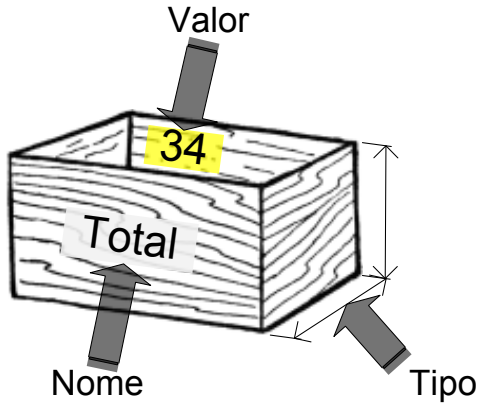


Figura 1 - Representação de uma variável

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.

1.3. NOME DAS VARIÁVEIS

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

1. O nome de uma variável pode ter um ou mais caracteres;
2. Nomes de variáveis ou funções no Octave devem começar com uma letra, ou com um dos seguintes caracteres especiais: '%', '_', '#', '!', '$', '?'.
3. Os próximos caracteres podem ser letras ou um dos seguintes caracteres especiais: '_', '#', '!', '$', '?'
4. Nomes podem ser tão longos quanto se queira, mas apenas os primeiros 24 caracteres serão levados em consideração.
5. Letras maiúsculas e minúsculas são diferentes.
6. Não poderão ser utilizados outros caracteres diferentes de letras, números ou sublinhado;

1.4. VARIAVÉIS NO OCTAVE

Para utilizar variáveis no Octave, precisamos apenas especificar o nome e o valor:

```
>>a = 1;

>> b = 2
b =  2
```

Perceba a diferença quando se utiliza o sinal ‘;’. Quando ele é inserido, o Octave atribui o valor a variável normalmente, a retirada do sinal irá fazer com que o Octave atribua o valor igualmente a opção anterior e imprima o valor no terminal.

Realizar a soma de variáveis e atribuir para uma nova variável é simples:

```
>>c = a + b;

>> c
c =  3

>> d = a + a
d =  2
```

Perceba que para verificar o valor de uma variável, basta apenas digitar o nome da variável (lembre-se que o sinal ‘;’ irá impedir que o valor seja exibido).

Caso queira, pode ser incluído valores diretamente:

```
>> 2 * 5
ans =  10

>> 2 * 5.1
```

```
ans =  10.200

>> 2 ^ 3
ans =  8

>> 2 / 5
ans =  0.40000
```

Observação: a utilização de espaço entre os valores e o sinal de operação é opcional.

Podemos também apresentar os valores de variável através do comando *disp*:

```
>>a = 3;

>> disp(a);
3
```

E uma situação especial e quando não definimos a variável que irá ser armazendo o resultado, nesse caso o Octave atribui automaticamente o resultado para variável *ans*:

```
>>5 * 3;

>> disp(ans);
15
```

1.5. VARIÁVEIS PRÉ-DEFINIDAS

Algumas variáveis já possuem valores pré-definidos:

Constante	Detalhes
I, i, J, j	Números imaginários puros
Inf, inf	Infinito
NaN, nan	Not a number. É o resultado de operações como '0/0 ou 'Inf - Inf'
SEEK_SET	Comandos opcionais da função fseek
SEEK_CUR	
SEEK_END	
eps	Retorna a precisão do computador
pi	A razão entre a circunferência de um círculo e seu diâmetro. Internamente ele é computador por `4.0 * atan (1.0)'.
realmax	O maior número em ponto flutuante que pode ser representado
realmin	O menor número em ponto flutuante que pode ser representado
stdin	Retornar os números dos arquivos correspondente a entrada padrão, saída padrão e saída de erro padrão.
stdout	

stderr	
e	Número de Euler ou Constante de Néper. É a constante que resolve a equação: log (e) = 1

```
>> pi
ans =  3.1416

>> e
ans =  2.7183

>> realmax
ans =   1.7977e+308
```

1.6. TIPOS DE VARIÁVEIS

Apesar de transparentes, as variáveis possuem tipos para fins de armazenamento.

1.6.1. Inteiros

Em grande parte das linguagens de programação, números naturais (0, 1, 2, 3, etc.) são conhecidos como inteiros sem sinal.

Para valores inteiros pode ser definidos a quantidade de bits utilizados para armazenamento do valor.

- Um inteiro **com** sinal de **n** bits tem uma faixa de valores de -2^{n-1} até $2^{n-1} - 1$
- Um inteiro **sem** sinal de **n** bits tem uma faixa de valores de 0 até $2^{n} - 1$

Por exemplo, se forem utilizados 8 bits para armazernar um inteiro, a escala de valores vão:

- Inteiro **com** sinal: de -2^{7} (-128) até $2^{7} - 1$ (127)
- Inteiro **sem** sinal: de 0 até $2^{8} - 1$ (255)

y=int8(x)	Um inteiro **com** sinal de 8-bits entre $[-2^{7}, 2^{7} - 1] = [-128, 127]$
y=uint8(x)	Um inteiro **sem** sinal de 8-bits entre $[0, 2^{8} - 1] = [0, 255]$
y=int16(x)	Um inteiro **com** sinal de 16-bits entre $[-2^{15}, 2^{15} - 1] = [-32768, 32767]$
y=uint16(x)	Um inteiro **sem** sinal de 16-bits entre $[0, 2^{15} - 1] = [0, 65535]$
y=int32(x)	Um inteiro **com** sinal de 32-bits entre $[-2^{31}, 2^{31} - 1] = [-2147483648, 2147483647]$
y=uint32(x)	Um inteiro **sem** sinal de 32-bits entre $[0, 2^{32} - 1] = [0, 4294967295]$

1.6.2. Ponto flutuante

O ponto flutuante é uma forma de representar números na forma racional (envolvendo frações). A padronização dos números em ponto flutuante em 1985, garantiu a homogeneidade da representação e cálculos desse tipo de número. Antes disso, cada fabricante adotava uma forma própria de representar e também de operar, muitas vezes o fator precisão era deixado em segundo plano em favor de um melhor desempenho dos cálculos.

Em Octave as variáveis por padrão são double, um tipo de ponto flutuante de 64 bits, mesmo quando o número é matematicamente um inteiro. Na prática uma variável double pode armazenar valores inteiros entre -2^{52} e 2^{52}.

Para números reais, o double consegue armazernar valores entre $2{,}22507385850720 \times 10^{-308}$ e $1{,}7976931348623157 \times 10^{308}$.

```
>> a = 3 * pi
a =  9.4248

>> b = 1/3
b =  0.33333
```

O armazenamento em ponto flutuante tem problema de arredondamento, verifique o resultado da equação abaixo:

```
>> 0.5 - 0.4 -0.1
ans =   -2.7756e-17
```

Outro problema que pode acontecer é extrapolar o limite de armazenamento:

```
>>printf("%25d\n", 2^53 + 1);
9007199254740992

>>printf("%25d\n", 2^53 + 2);
9007199254740994

>>printf("%25d\n", 2^53 + 3);
9007199254740996.
```

Obs.: O comando `printf` será visto em capítulo a parte, mas para resumir, ele é um comando que permite especificar com mais detalhes como será apresentada a saída.

1.6.3. String

Strings são codificadas como uma sequência de caracteres terminada pelo carácter nulo (valor 0). Cada carácter é um número inteiro convertido através de uma definição para um carácter. A tabela mais utilizada é a ASCII, que apresenta a equivalência de vários símbolos para o binário.

Strings são delimitadas pelo sinal de aspas duplas (“ e ”) , e a concatenação (junção de strings) pode ser feito utilizando os colchetes: “[ ]”

```
>>x = "aula";

>>y = "programação";

>> [x y]
ans = aulaprogramação
```

1.6.4. Tipos de variáveis dinâmicos

Quando nos criamos e gerenciamos variáveis, Octave permite alterar o tipo da variável dinamicamente. Isto significa que podemos criar um valor real e logo em seguida alterar o conteúdo da variável para outro tipo:

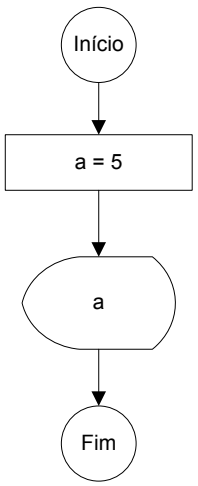
```
>> a = 5.1
a =  5.1000

>> a = "valor"
a = valor
```

1.7. LENDO E IMPRIMINDO VALORES NO OCTAVE

Além do comando *disp*, o Octave possui o comando printf, um comando mais amplo para imprimir valores.

1.7.1. Imprimindo valores

A impressão de valores através do comando printf:




```
>>a = 5;

>> printf("O valor de a = %d\n", a);
O valor de a = 5
```

Figura 2 - Imprimindo valores

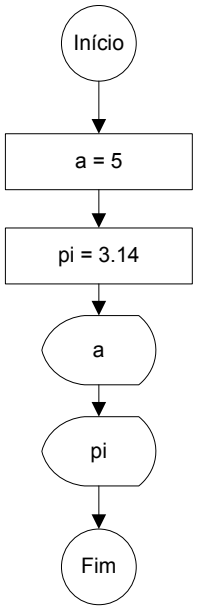
Perceba que o *printf* possui alguns caracteres estranhos dentro das aspas, o primeiro %d se refere ao tipo de dado esperado, outros formatos que poderão ser utilizados:

Tabela 1 - Caracteres coringas

Caractere	Descrição
%d ou %i	Número decimal inteiro.
%f	Número real (ponto flutuante)
%s	String

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:




```
>>a = 5;

>>pi = 3.14;

>> printf("A = %d, PI = %f\n", a, pi);
A = 5, PI = 3.140000
```

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:

```
>>a = 5;

>>pi = 3.14;

>>printf("A = %d, PI = %f\n", a, pi);
A = 5, PI = 3.140000

>>printf("Resultado 1: %d\n", a * pi);
Resultado 1: 15
```

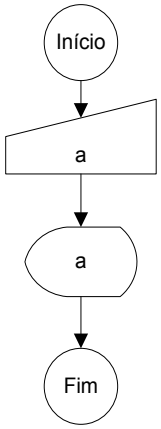
```
>>printf("Resultado 2: %f\n", a * pi);
Resultado 1: 15.700000
```

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

```
Resultado 1: 15
Resultado 2: 15.700000
```

1.7.2. Lendo um valor

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

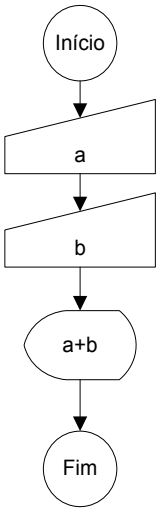



```
>>a = input("Digite um valor: ");
Digite um valor: 4

>>printf("Valor digitado: %d\n", a);
Valor digitado: 4
```

Figura 4 - Lendo um valor

1.8. 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:




```
>>a = input("Digite um valor de A: ");
Digite um valor de A: 5

>>b = input("Digite um valor de B: ");
Digite um valor de B: 3

>>printf("Resultado: %d", a + b);
Resultado: 8
```

Figura 5 - Algoritmo para somar dois valores

1.9. 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
   b. média_final
   c. R$
   d. int
   e. 1dia
   f. teste 1
   g. area.do.quadrado
   h. valor real
   i. a+b

1.10. EXERCÍCIOS PROGRAMAÇÃO

1. Faça um fluxograma/programa que receba dois valores e apresente a divisão entres.
2. Faça um fluxograma/programa que receba dois valores, um é a base e outro o expoente e apresente o resultado da potência
3. Um fluxograma/programa que receba o valor do raio de um círculo e imprima a área.
4. O Sr. José do mercadinho da esquina solicitou que você faça um fluxograma/programa para agilizar o cálculo do custo das maças. Ele vende maça ao valor de R$10,00 por dúzia. O cliente pode comprar em fração de meia dúzia ou dúzias completas. Seu programa deve receber o valor de dúzias e apresentar o resultado.

Exemplos:

```
Digite a quantidade de dúzias: 4
Valor das maças = R$ 40.000000

Digite a quantidade de dúzias: 1.5
Valor das maças = R$ 15.000000
```

5. Faça um fluxograma/programa que receba os valores dos catetos e calcule a hipotenusa de um triângulo retângulo (para calcular a raiz utilize a função sqrt(valor)).

Exemplos:

```
Digite o valor do Cateto a: 2
Digite o valor do Cateto b: 4
O valor da hipotenusa é 4.472136

Digite o valor do Cateto a: 9
Digite o valor do Cateto b: 12
O valor da hipotenusa é 15.000000
```

2. SISTEMAS DE NUMERAÇÃO

Utilizar a notação decimal é interessante para nós seres humanos, principalmente pela associação com o número de dedos. Porém para o computador a manipulação de dados através dessa notação é atualmente inviável, e por isso, o computador utiliza a representação de informação através da notação binária (dois estados: ligado ou desligado, 0 ou 1).

2.1. SISTEMA DE NUMERAÇÃO DECIMAL

Estamos tão acostumados com a representação de valores no formato decimal que passa despercebido a forma como são compostos, por exemplo, o número 456 é a combinação de:

6 x unidades (peso 1 ou 10^0)
5 x dezenas (peso 10 ou 10^1)
4 x centenas (peso 100 ou 10^2)

Ou outra forma de representação é através do peso das casas:

Valor					4	5	6
Peso	10^6	10^5	10^4	10^3	10^2	10^1	10^0
Resultado					400	50	6

Somando-se os resultados:

$$400 + 50 + 6 = 456$$

Parece uma brincadeira, mas lembre-se que já estamos acostumados com esse sistema de numeração e já sabemos como representar ele.

2.2. SISTEMA DE NUMERAÇÃO BINÁRIO

A codificação utiliza pelo computar possui apenas dois estados possíveis, com isso a tabela de pesos utiliza a base 2, e como podemos ter apenas os valores 0 e 1 nas casas, vamos fazer a demonstração de quanto vale o número 1110010_2 (perceba o 2 subscrito após o número, ele representa em qual base se está apresentado o número)

Valor	1	1	1	0	0	1	0
Peso	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Resultado	64	32	16	0	0	2	0

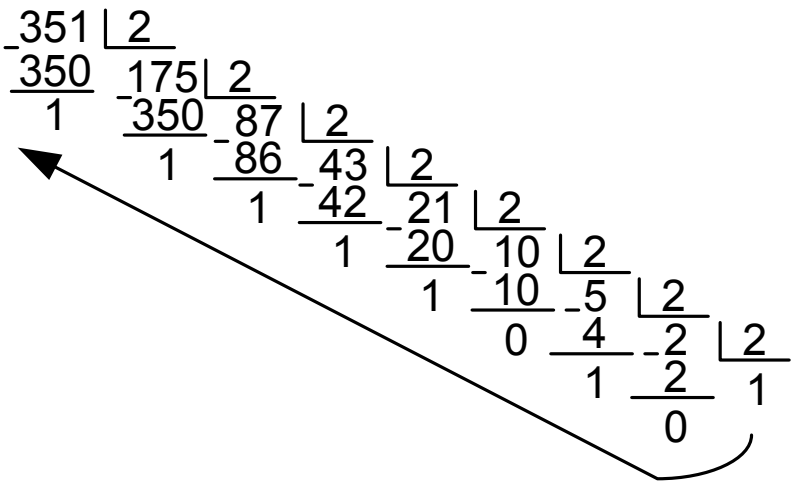
Somando os resultados:

$$64 + 32 + 16 + 0 + 0 + 2 + 0 = 114_{10}$$

2.2.1. Convertendo de decimal para binário

O processo de conversão de decimal para binário pode ser feito de duas formas, de forma a exemplificar os processos vamos converter o número 351_{10} para binário:

Método 1 – Dividindo o número por 2 sucessivamente



Utilizando os resultados de trás para frente:

$$351_{10} = 101011111_2$$

Método 2 – Subtração do peso das casas

Nesse método, preciso verificar se posso subtrair o valor do número pelo peso da casa, caso positivo, acrescento 1 (um) a saída e o resto da subtração é enviada para a próxima casa. Se o número é menor que o peso da casa, acrescento 0 (zero) para a saída e continuo o mesmo número no valor da próxima casa.

Valor		351–256=	95	95-64=	31	31-16=	15-8=	7-4=	3-2=	1-1=0
Peso	$2^9 = 512$	$2^8 = 256$	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Saída		1	0	1	0	1	1	1	1	1

Novamente chegamos ao resultado:

$$351_{10} = 101011111_2$$

2.3. SISTEMA DE NUMERAÇÃO OCTAL

Para nós humanos, entender uma sequência de 0 e 1 é confuso e fazer a conversão de binário para decimal exige certo cálculos para se obter o valor, de forma a simplificar a visualização de sequências de bits são utilizados outros métodos e sistema de numeração. Um dele é o octal, que utiliza os número de 0 a 7, totalizando 8 possíveis combinação por casa.

2.3.1. Conversão de octal para decimal

Para converter um número em octal para decimal utilizamos o processo de casas e pesos, por exemplo, para converter o número 753_8 para decimal fizemos:

Valor				7	5	3
Peso	$8^5 = 32.768$	$8^4 = 4096$	$8^3 = 512$	$8^2 = 64$	$8^1 = 8$	$8^0 = 1$
Resultado				7x64=448	5x8=40	3x1 = 3

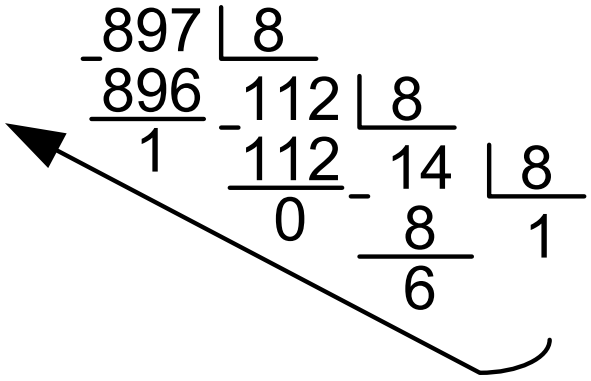
O resultado será:

$$448 + 40 + 3 = 491_{10}$$

2.3.2. Conversão de decimal para octal

Para converter um número em decimal para octal, utilizamos o processo de divisão sucessivas por 8.

Veja um exemplo, para converter o número 897_{10} para octal



O resultado é 1601_8

2.3.3. Convertendo de binário para octal

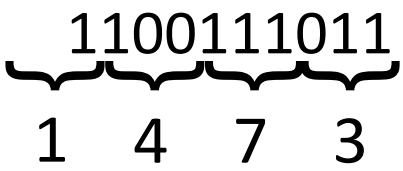
O real motivo de se utilizar o sistema de numeração octal deve-se a forma de representar os números em binário. Por isso o sistema de conversão é simples, porém antes e interessante conhecer o número em octal e se equivalente em binário.

Tabela 2 - Equivalência octal-binária

Octal	Binário
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Com base nessa tabela, basta apenas agrupar a cada três bits o número em binário e verificar seu equivalente em octal, caso necessário podemos preencher com zeros a esquerda nosso número em binário.

Veja o exemplo, para converter o número 1100111011_2 para octal:

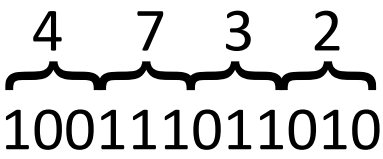


Resultado: $1100111011_2 = 1473_8$

2.3.4. Conversão de octal para binário.

Para a conversão de octal para binário o processo é o inverso daquele apresentado anteriormente, utilizando a Tabela 2 utiliza-se a equivalência do número em octal

Para exemplificar, vamos converter o número 4732_8 para binário:



O resultado da conversão: 100111011010_2

2.4. SISTEMA DE NUMERAÇÃO HEXADECIMAL

De uma forma parecida ao sistema octal, o hexadecimal é utilizado para melhor visualizar uma sequencia de bits, sendo inclusive muito mais utilizado que o octal. De forma a completar a lista com 16 símbolos, após o 9 se inicia a utilização de letras, começando no A até o F, essa equivalência é vista na Tabela 3

Tabela 3 - Equivalência Decimal – Hexadecimal

Decimal	Hexadecimal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

2.4.1. Conversão de hexadecimal para decimal.

Para a conversão utilizamos o sistema de casas agora com a base 16. Para exemplo, vamos converter o número A5D (perceba a substituição da letra pela seu equivalente decimal)

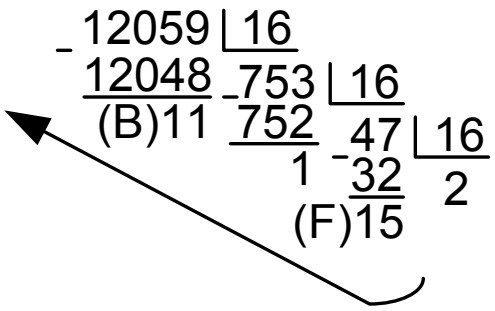
Valor			A	5	D
Peso	$16^4 = 65536$	$16^3 = 4096$	$16^2 = 256$	$16^1 = 16$	$16^0 = 1$
Resultado			10x256=2560	5x16=80	13x1 = 13

Resultado = 2560 + 80 + 13 = 2653_{10}

2.4.2. Conversão de decimal para hexadecimal

O processo ocorre por divisão, sempre cuidando para o caso do resto for maior que 10 deve ser substituído pelo sua letra equivalente.

Convertendo o número 12059_{10} para a base hexadecimal:



Resultado = $2F1B_{16}$

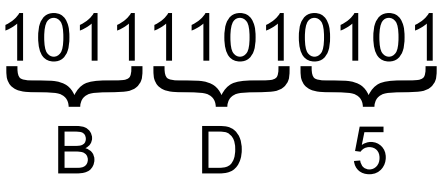
2.4.3. Conversão de hexadecimal para binário

A conversão entre essas bases segue o mesmo raciocínio da conversão entre octal e binário, porém com 4 casas, que é a quantidade de casas necessária para representar os 16 símbolos em hexadecimal.

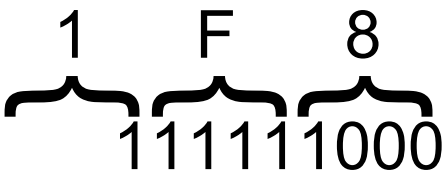
Hexadecimal	Binário	Hexadecimal	Binário
0	0000	**8**	1000
1	0001	**9**	1001
2	0010	**A**	1010
3	0011	**B**	1011
4	0100	**C**	1100
5	0101	**D**	1101
6	0110	**E**	1110
7	0111	**F**	1111

A conversão acontece agrupando os binário em grupo de 4 e verificando seu equivalente em hexadecimal, o processo inverso também é válido.

Por exemplo, para converter o número 101111010101_2 para hexadecimal:



E de $1F8_{16}$ para binário:



2.5. ENDEREÇAMENTO E ORDEM DE BYTES

Os programas que utilizam vários bytes para armazenar dados, precisar de duas informações: qual o endereço de armazenamento e qual a ordem dos bytes na memória. Para dados armazenados em vários bytes, geralmente uma sequência de bytes contínuos é reservado para esse dado. O endereço dado é o byte inicial, por exemplo, se uma variável *x* é armazenada no endereço *0x100*, o valor da expressão *&x* é *0x100*. Portanto os quatro bytes da variável *x* são armazenados nos endereços *0x100*, *0x101*, *0x102* e *0x103*.

A ordem dos bytes é a forma como os dígitos serão armazenamentos no espaço de memória, podendo iniciar pelo mais significativo (big endian) ou pelo menos significativo (little endian)[1]. Exemplos de arquitetura que são little endian incluem: x86 (inclusive x86-64), 6502 (inclusive 65802, 65C816), Z80 (inclusive Z180, eZ80 etc.), MCS-48, 8051, DEC Alpha, Altera Nios, Atmel AVR, SuperH, VAX, e PDP-11.

Nossa forma de representar número é equivalente ao big endian, onde começamos a representar um número com o maior peso primeiro e a esquerda. Algumas arquitetura big endian são: Motorola 6800 e 68k, Xilinx Microblaze, IBM POWER, System/360 e seus sucessores System/370, ESA/390, e z/Architecture.

Ainda há um conjunto de arquitetura que a ordem dos bytes pode configurado no momento de inicialização, denominadas bi-endian. São exemplo de arquitetura bi-endian ARM, PowerPC, Alpha, SPARC V9, MIPS, PA-RISC e IA-64.

A seguir segue um exemplo de armazenamento do valor 0x0A0B0C0D nas formas Big e Little Endian:

0x103	0D
0x102	0C
0x101	0B
0x100	0A

Figura 6 - Big Endian

0x103	0A
0x102	0B
0x101	0C
0x100	0D

Figura 7 - Little Endian

2.6. REPRESENTANDO STRING

String em C é codificado como uma sequência de caracteres terminada pelo caracter nulo (valor 0). Cada caracter é um número inteiro convertido através de uma definição para um caracter. A tabela mais utilizada é a ASCII, que apresenta a equivalência de vários símbolos para o binário, por exemplo a string “arquitetura” será representada conforme mostra a Figura 8, perceba que o computador irá armazenar o valor em binário.

Caracter	a	r	q	u	i	t	e	t	u	r	a
Decimal	97	114	113	117	105	116	101	116	117	114	97

[1] Protocolos seriais também são classificados quanto a ordem de envio das informações. USB, RS-232, RS-422 e RS-485 são exemplos de padrões seriais do tipo little

Binário	1100001	1110010	1110001	1110101	1101001	1110100	1100101	1110100	1110101	1110010	1100001
Hexadecimal	61	72	71	75	69	74	65	74	75	72	61

Figura 8 - Representação de caracteres

2.7. REPRESENTAÇÃO DE INTEIROS

Existem diversas formas de representar um número inteiro, a princípio o computador é capaz de armazenar inteiros positivos apenas, porém, como proceder se precisar armazenar um número negativo? Esse capítulo irá tratar dessas formas de representação

2.7.1. Codificação de inteiros sem signal

Assumimos que armazenamos um inteiro com a largura ω, o vetor que contêm os valores chamaremos de ň, uma sequência de bits que será convertido para inteiro sem sinal. A equação que transcreve a forma de conversão é apresentado a seguir:

$$I_{ss} = \sum_{i=0}^{\omega-1} \check{n}[i]\times 2^i$$

2.7.2. Codificação de inteiros com signal[2]

Para representar um número com sinal o processo mais comum é o complemento de 2. Ele utiliza o bit mais significativo da palavra como sinal. A expressão que define a forma de conversão é:

$$I_{cs} = -(\check{n}[\omega-1]\times 2^{\omega-1}) + \sum_{i=0}^{\omega-2} \check{n}[i]\times 2^i$$

Por exemplo, qual o número em decimal representado pelo número 11011010_2, perceba que a largura do vetor é 8:

Valor	1	1	0	1	1	0	1	0
Peso	$2^7 = 128$	$2^6 = 64$	$2^5 = 32$	$2^4 = 16$	$2^3 = 8$	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$
Resultado	128	64	0	16	8	0	2	0

- (128) + 64 + 16 + 8 + 2 = -38

2.8. PONTO FLUTUANTE

[2] Fim do mundo para o Linux e similares, um problema de limitação na variável de armazenamento da data nos sistemas com padrão POSIX podem causar problemas, o fim será as 03hrs14min07seg de 19 de Janeiro de 2038. Para maiores detalhes procure por “Bug 2038”

O ponto flutuante é uma forma de representar números na forma racional (envolvendo frações). A padronização dos números em ponto flutuante em 1985 garantiu a homogeneidade da representação e cálculos desse tipo de número. Antes disso, cada fabricante adotava uma forma própria de representar e também de operar, muitas vezes o fator precisão era deixado em segundo plano em favor de um melhor desempenho dos cálculos.

2.8.1. Números fracionários binários

Na nossa forma norma de representação de números fracionários decimais, utilizamos o sistema de peso também no lado direito da vírgula que separa a parte inteira da fracionária. Por exemplo, o numero 12,345 seria representado assim:

Valor		1	2	3	4	5	
Peso	...	10^1	10^0	10^{-1}	10^{-2}	10^{-3}	...
Resultado		10	2	0,3	0,04	0,005	

O resultado seria a soma dos resultados, ou na forma matemática:

$$D = \sum_{i=-n}^{m} 10^i \times d[i]$$

Onde *D* seria o número decimal final, *m* a quantidade de casas inteiras, *n* o número de casas fracionárias, e *d[i]* a parte do número decimal na casa *i*.

Por analogia, considere agora essa forma para representar o sistema binário, com o mesmo princípio de casas e pesos. Por exemplo, vamos representar o número 101.11_2

Valor		1	0	1	1	1	
Peso	...	$2^2 = 4$	$2^1 = 2$	$2^0 = 1$	$2^{-1} = 1/2$	$2^{-2} = 1/4$	...
Resultado		4	0	1	1/2	1/4	

$$\text{Resultado } = 5\frac{3}{4}$$

A representação matemática é:

$$B = \sum_{i=-n}^{m} 2^i \times b[i]$$

Porém essa forma de representar números fracionários tem problemas de precisão[3][4], por exemplo, para representar o número 0,2 (1/5), não há formas precisão de representação, conforme mostrado na Tabela 4.

Tabela 4 - aproximação de 1/5

Representação	Valor	Decimal
$0{,}01_2$	1/4	$0{,}25_{10}$
$0{,}0011_2$	3/16	$0{,}1875_{10}$
$0{,}001101_2$	13/64	$0{,}203125_{10}$
$0{,}00110011_2$	51/256	$0{,}19921875_{10}$

2.8.2. Representação de ponto flutuante de acordo com a IEEE

Com o padrão IEEE 754 de 1985, baseada num pequeno e consistente conjunto de princípios tornou a representação em número em ponto flutuante elegante e entendível.

A IEEE 754 representa um número no formato $V = (-1)^S \times M \times 2^e$ onde:

- S: sinal do número, negativo é definido por S = 1 ou positivo S = 0
- M: Mantissa, é a representação da parte fracionária
- e: expoente da base.

A IEEE 754 padroniza o formato em dois tamanhos, o ponto flutuante de **Precisão simples** e o de **Precisão dupla**. Cada um com diferentes tamanhos de campos conforme apresentado respectivamente nas figuras Figura 9 Figura 10

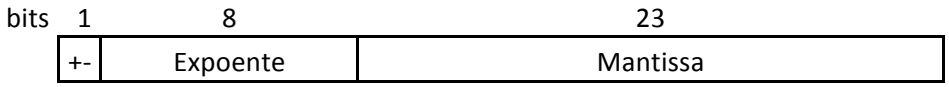


Figura 9 - Ponto flutuante de precisão simples

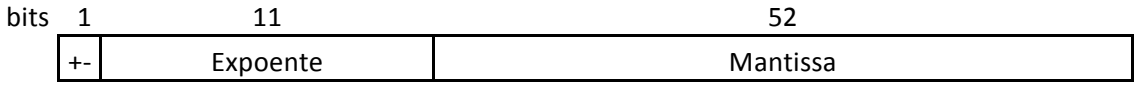


Figura 10 - Ponto flutuante de precisão dupla

O expoente é definido através do calculo com o *bias*, esse número é obtido através do número de bits no expoente com a fórmula $2^{k-1} - 1$. Para o caso de precisão simples, o bias é $2^{8-1} - 1 = 127$. Para o ponto flutuante de precisão dupla, $2^{11-1} - 1 = 1023$.

[3] Um fato curioso sobre essa problema de representação foi na guerra do Golfo, para saber mais detalhes pesquise num site de busca por "erro arredondamento míssil Patriot".

[4] Ainda podemos verificar problemas de arredondamento no Excel, para isso numa célula qualquer insira a equação = (0,5-0,4-0,1). O resultado deveria ser 0.

2.8.3. Convertendo um número fracionário decimal para binário[5]

De forma a exemplificar a conversão, vamos considerar uma definição própria do tamanho do ponto flutuante, o sinal será de 1 bit, o expoente de 3 bits (bias igual a 3) e mantissa de 4 bits, totalizando 8 bits para armazenar o número que será nosso exemplo: 2,625:

1. Para a parte inteira, realiza-se a conversão normal de decimal para binário: $2_{10} = 10_2$
2. Para a parte fracionária, faz-se sucessivas multiplicações da parte fracionária por 2:

   0,625 × 2 = 1,25 [1] *Retira o número inteiro e continua a conta*
   0,25 × 2 = 0,5 [0] *Gerou 0, continuando...*
   0,5 × 2 = 1,0 [1] *Gerou 1 e restou 0, para aqui.*

   Portanto, $0{,}625_{10} = 0{,}101_2$

3. Juntando a parte inteira e a parte fracionária, temos 10.101_2
4. Adicionando o expoente: $10.101_2 \times 2^0$
5. Normalizando: $10.101_2 \times 2^0 = 1.0101_2 \times 2^1$ (verifique que foi deslocado o ponto decimal uma casa para a direita, por isso, o expoente aumentou uma casa).
6. Mantissa: 0101
7. Expoente: $1 + 3 = 4_{10} = 100_2$
8. Bit de sinal positivo: 0
9. Resultado é 01000101, em hexadecimal 45_{16}

Vamos para outro exemplo, converter o número -39887.5625 para o formato IEEE 754 de precisão simples:

1. A parte inteira: $39887_{10} = 1001101111001111_2$
2. A fração $0{,}5625_{10} = 0{,}1001_2$

0,5625 × 2 = 1,125 [1] *Generate 1 and continue with the rest.*
0,125 × 2 = 0,25 [0] *Generate 0 and continue.*
0,25 × 2 = 0,5 [0] *Generate 0 and continue.*
0,5 × 2 = 1,0 [1] *Generate 1 and nothing remains.*

3. A conversão completa: $39887{,}5625_{10} = 1001101111001111{,}1001_2$
4. Normalizando: $1001101111001111{,}1001_2 = 1{,}0011011110011111001_2 \times 2^{15}$.
5. Mantissa é 00110111100111110010000, o expoente é 15 + 127 = 142 = 100011102, bit de sinal é 1.
6. Portanto 39887,5625 é 11000111000110111100111110010000 = $C71BCF90_{16}$

2.8.4. Convertendo um número fracionário binário para decimal.

O processo inverso, para converter $C4A42A00_{16}$ para decimal, através da normativa IEEE de precisão simples:

[5] Forma de conversão retirado de http://sandbox.mc.edu/~bennet/cs110/flt/dtof.html

1. Convertendo: $C4A42A00_{16} = 11000100101001000010101000000000_2$
2. Sinal: 1 = negativo
3. Expoente: $10001001_2 = 137 – 127$ (bias) = 10
4. Mantissa: $01001000010101000000000_2 = 0,28253173828125_{10}$
5. Aplicando a mantissa e o expoente = $1,28253173828125 \times 2^{10} = -1313,3125_{10}$

3. 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

3.1. 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 pela direita	/
Divisão pela esquerda	\
Potência	^ ou **
Transpor	‘

Há dois operadores para divisão, um que segue o sentido normal como estamos acostumados:

```
>>2 / 4
 ans  =

    0.5
```

E outro que faz o processo com os operadores invertidos:

```
>>2 \ 4
 ans  =

    2.
```

Para a potência, podem ser utilizados tanto o sinal ‘^’ quanto ‘**’:

```
>>2 ^ 2
 ans  =

    4.

>>2 ** 2
 ans  =

    4.
```

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
```

3.2. ESTADOS LÓGICOS

Em várias linguagem, os operadores relacionais e lógicos só podem retornar dois estados possíveis Verdadeiro ou Falso (em inglês *True* ou *False*) representado pelos números 1 e 0 respectivamente.

Porém, qualquer valor diferente de 0 (zero) será considerado verdadeiro.

Veja alguns exemplos:

```
>> 0 & 1
 ans = 0

>>-1 & 2
 ans  = 1

>>1 & 2
 ans  = 1
```

3.3. 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 (Verdadeiro ou Falso). 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	==
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

3.4. 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 Octave
e	and	**&**
ou	or	**|**
não	not	~ ou !

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

3.5. 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 >= (A + 2) ( )
c) C == (B – A) ( )
d) (B + A) <= 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 <= D) ( )
b) (A+B) > 10 **|** (A+B) == (C+D) ( )
c) (A>=C) **&** (D >= 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.

Área = (Base * 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.

Segundos = (Quantidade de horas * 3600) + (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

Consumo médio = distância total / 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.

4. FUNÇÕES

Funções são formas de operar elementos de forma mais natural e são muito utilizadas para melhorar estrutura um programa.

Uma função começa com a palavra chave **function** e termina com **endfunction** veja um exemplo:

```
>>function res=soma(a, b)
>>res = a + b;
>>endfunction

>>soma(4, 5)
 ans = 9
```

Detalhando as linhas

Na linha `function res=soma(a, b)` começamos uma função chamando a palavra **function**, logo em seguinda a variável **res** será responsável por pelo retorno de dados para o local em que foi chamado. **Soma** se refere ao nome da função e **a** e **b** os valores que serão passados.

Dentro da função, a variável de retorno **res** irá receber o resultado da soma e ao final da função irá ter seu valor retornando a função que chamou.

A linha `soma(4, 5)` irá chamar a função correspondente, passando os valores 4 e 5, ao final apresentado o resultado.

A sintaxe padrão da função é a seguinte:

```
Function [<retorno>]=<Nome_Função>(<lista de argumento>)
     <Processamento>
endfunction
```

O retorno pode ser uma única variável ou um conjunto de variáveis, nesse caso a lista de variáveis de retorno devem ser inseridas dentro de colchetes (“[“ e “]”)

Perceba no próximo exemplo, que a função processa retorna dois valores, um com o resultado da soma dos parâmetros e outro com a subtração:

```
>>function [soma, subtrai]=processa(a, b)
>>soma = a + b;
>>subtrai = a - b;
>>endfunction

>>[res_soma, res_sub] = processa(6, 4)
 res_soma =  10
```

```
res_sub =  2
```

4.1. EXERCÍCIOS

1. Escreva uma função que calcule e retorne a distância entre dois pontos (x1, y1) e (x2, y2).
2. Escreva uma função *potencia(base, expoente)* que, quando chamada, retorna $base^{expoente}$. Por exemplo, potencia(3, 4) deve retornar 81. Assuma que expoente é um inteiro maior ou igual a 1.
3. Crie funções que recebam um valor de x e retorne o resultado das seguintes equações matemáticas:
    a. $x + 2$
    b. $x – 2*x$
    c. $x^2 – 4$
    d. $x^3 + 2x^2 - 6x + 2$
4. Faça um programa que receba os valores de **a, b** e **c** e através da formula de Bhaskara, retorne as duas raízes da expressão.

5. 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.

5.1. 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.

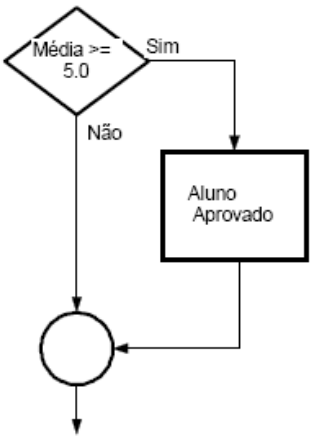
A decisão é sempre composta por um ou mais operador relacional, e quando mais de um operador relacional estiver na condição, obrigatoriamente deve-se utilizar o operador lógico para conectar as expressões.

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:




```
media = input("Digite a nota: ")

if media >= 5
    printf("Aprovado\n");
 end
```

Figura 11 - Comando IF

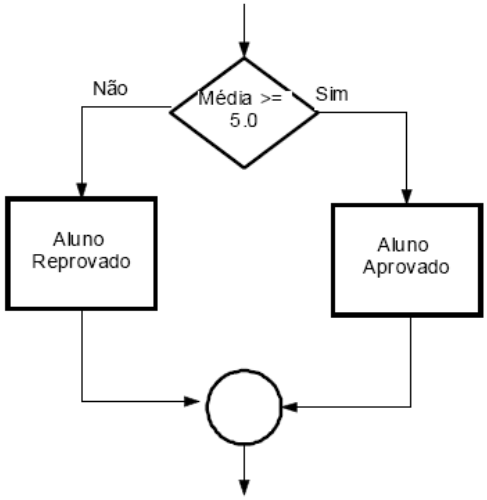
5.2. SE... SENÃO / IF....ELSE

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 Octave




```
media = input("Digite a nota: ")

if media >= 5
    printf("Aprovado\n");
else
    printf("Reprovado\n");
end
```

Figura 12 - 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”.

5.2.1. Exercício

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

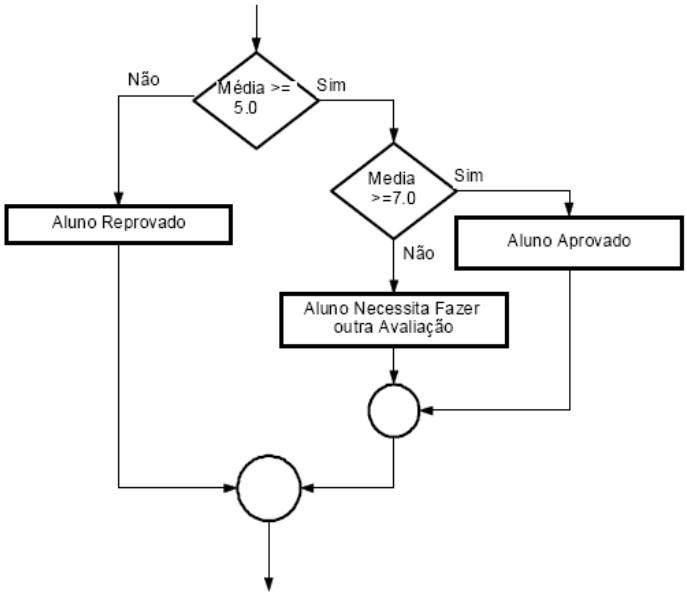
5.3. IF/ELSE ENCADEADOS

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".

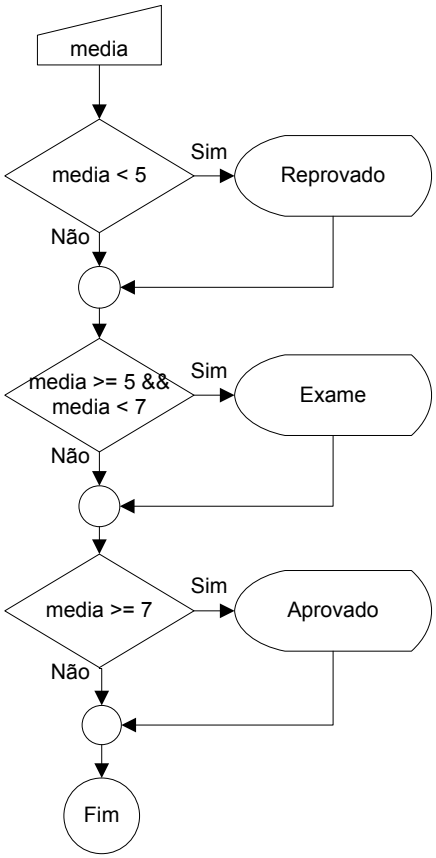



```
media = input("Digite a nota: ")

if media >= 5
    if media >= 7
        printf("Aprovado");
    else
        printf("Exame");
    end
else
    printf("Reprovado");
end
```

Figura 13 - IFs encadeados

5.4. VÁRIOS IFS

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




```
media = input("Digite a nota: ")

if media < 5
    printf("Reprovado");
elseif media >= 5 & media < 7
    printf("Exame");
elseif media >= 7
   printf("Aprovado");
end
```

Figura 14 - 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.

5.5. SWITCH... CASE

A estrutura de decisão SWITCH/CASE 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:

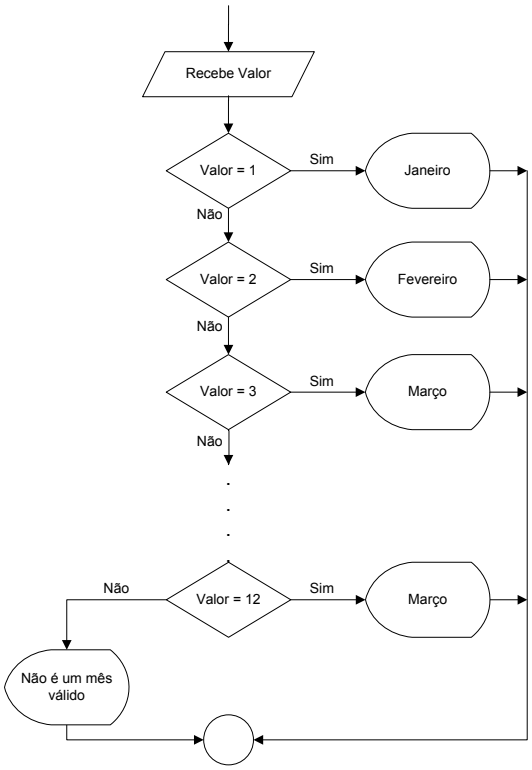


Figura 15 - Diagrama de bloco do Switch

Em Octave:

```
mes = input("Digite o mês:");

switch (mes)
    case 1
        printf("Janeiro\n")
    case 2
        printf("Fevereiro\n")
    case 3
        printf("Março\n")
    case 4
        printf("Abril\n")
    case 5
        printf("Maio\n")
    case 6
        printf("Junho\n")
    case 7
        printf("Julho\n")
    case 8
        printf("Agosto\n")
    case 9
        printf("Setembro\n")
    case 10
        printf("Outubro\n")
    case 11
```

```
        printf("Novembro\n")
    case 12
        printf("Dezembro\n")
    otherwise
        printf("Mês inválido\n")
end
```

Figura 16 - Código fonte do switch

Outra forma de usar o comando Switch/case é agrupando as opções e caso seja uma dela, o comando será executado:

```
valor = input("Digite um valor:");

switch (valor)
    case {1, 2, 3}
        printf("Valor digitado foi 1, 2 ou 3\n")
    case {4, 5, 6}
        printf("Valor digitado foi 4, 5 ou 6\n")
    otherwise
        printf("Outro valor\n")
end
```

5.6. 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 retorno o valor que deve ser pago.

2. Faça um programa que leia um número inteiro e mostre uma mensagem indicando se é positivo, negativo ou zero.

3. Crie um programa que receba um número inteiro e classifique-o em par ou ímpar. Dica: a função modulo(A, B) retorna o resto da divisão do número A pelo B, por exemplo: modulo(5, 2) irá apresentar 1, enquanto modulo(6, 2), apresentará 0.

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

6. VETORES E MATRIZES

Vetores e matrizes são construções especiais de variáveis, são um agrupamento de variáveis do mesmo tipo em que seus elementos são acessados através de índices. O Octave foi programado para lidar com vetores e matrizes de forma eficiente e elegante.

6.1. CRIANDO VETORES E MATRIZES

Para criar um vetor ou matriz é necessário conhecer alguns símbolos básicos:

- Colchetes (“[" e “]“): marcam o início e o fim de uma matriz;
- Vírgula (“,“): separa os valores em colunas diferentes (também pode ser utilizado o espaço (“ “)para separar os elementos;
- Ponto e vírgula (“;”): separa os valores de diferentes linhas.

Exemplos de criação de vetores:

```
     >>v1 = [1, 2, 3] % Vetor com uma linha e três
colunas, elementos separados por vírgula
      v1  =

         1.    2.    3.

     >>v2 = [4 5 6] % Vetor com uma linha e três
colunas, elementos separados por espaço
      v2  =

         4.    5.    6.

     >>v3 = [1; 2; 3] % Vetor com três linhas e uma
coluna
      v3  =

         1.
         2.
         3.
```

Exemplos de criação de matrizes:

```
     >>m1 = [1, 2, 3; 4, 5, 6] % Matriz de 2 linhas e
3 colunas
      m1  =

         1    2    3
         4    5    6

     >>m2 = [4 5 6; 7 8 9] % Matriz de 2 linhas e 3
colunas
      m2  =

         4    5    6
         7    8    9

     >>m3 = [10 11 12 % Criando uma matriz com quebra
de linhas
     >>13 14 15
     >>16 17 18]
      m3  =

         10    11    12
         13    14    15
         16    17    18
```

Ou pode-se criar uma matriz vazia:

```
     >>m4 = []
      m4 = [](0x0)
```

6.2. SELECIONANDO OU ALTERANDO O VALOR DE UM ELEMENTO

Para selecionar ou alterar um elemento da matriz utilizar os índices de linha e coluna, começando a contar pelo valor 1:

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
A =

1    2    3
4    5    6
7    8    9
10   11   12
13   14   15

>>A(1, 3) % Selecionar o elemento na linha 1,
coluna 3
 ans =  3

>>A(2, 1) = 99 % altera o elemento da linha 2,
coluna 1 para 99
 A  =

    1     2     3
    99    5     6
    7     8     9
    10    11    12
    13    14    15
```

6.3. MATRIZ E FUNÇÕES

Funções podem receber como parâmetros de entrada vetores e matrizes:

```
>>function y=soma2(x)
>>y=x+2;
>>endfunction

>>A = [1 2 3; 4 5 6]
 A  =

   1   2   3
   4   5   6

>>res = soma2(A)
 res  =

   3   4   5
   6   7   8
```

Outro exemplo:

```
>>function [r1, r2]=opera(x)
>>r1 = x + 2;
```

```
>>r2 = x – 2;
>>endfunction

>>A = [1 2 3; 4 5 6]
 A  =

   1   2   3
   4   5   6

>>[resSoma, resSub] = opera(A)
resSoma =

    3   4   5
    6   7   8

resSub =

   -1   0   1
    2   3   4
```

6.4. O OPERADOR ":"

O operador ":" é um operador com várias finalidades dentro do Octave

6.4.1. Criando sequências de elementos

No Octave, pode ser utilizado o operador ":" para criar uma sequência de elementos num vetor:

```
>>a = [1:10]
 a =

    1    2    3    4    5    6    7    8    9   10

>>a = [1:2:10]
 a  =

   1   3   5   7   9

>>a = [2:2:10]
 a  =

    2    4    6    8   10
>>a = [2:1] % Cria uma sequência começando em 2 e terminando em 1??
```

```
a = [](1x0)


>>a = [2:-1:1] % Para criar uma sequência
decrescente é necessário especificar o passo com -1
 a =

   2   1

>>a = [10:-1:1]
 a =

   10    9    8    7    6    5    4    3    2    1
```

A sintaxe de uso do operador : para a criação de vetores é o seguinte:

[A:B] Criar uma sequência entre os números A e B, se o valor de B for maior do que A, o vetor criado é vazio.

[A:X:B] Cria uma sequência começando em A, com intervalo X entre os elementos até o valor B, desde que a sequência não ultrapasse o valor B.

Esse comando também pode ser utilizado para criar uma sequência decrescente. Nesse caso o valor do incremento deve ser negativo e o valor de A deve ser maior do que B.

Pode-se criar vetores com linhas em cada uma como uma sequência diferente, porém deve-se atentar ao número de elementos em cada linha que deve ser igual:

```
>>b = [1:2:10; 2:2:10]
 b =

    1    3    5    7    9
    2    4    6    8   10

>>c = [1:2:10; 2:2:10; 1:3:15]
 c =

    1    3    5    7    9
    2    4    6    8   10
    1    4    7   10   13
```

6.4.2. Selecionando e alterando linhas, colunas ou elementos

O operador “:” pode ser utilizado para selecionar linhas, colunas ou elementos:

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
 A  =

    1    2    3
    4    5    6
    7    8    9
   10   11   12

>>B = A(:, 1)
 B  =

    1
    4
    7
    10
```

Nesse exemplo foi criado a matriz A, depois foi selecionado todas as linhas através do operador “:” e apenas a primeira coluna;

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.

>>B = A(2:4, 1)
 B  =

    4.
    7.
    10.
```

O comando pode ser utilizado para passar uma sequência de elementos, nesse caso foi utilizado para retornar entre as linhas e 2 e 4 apenas a primeira coluna

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.

>>B = a(2:4, 1:2)
```

```
 B  =

    4.     5.
    7.     8.
    10.    11.
```

Retorna as linhas 2 a 4 e as colunas 1 e 2.

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    13.    14.    15.

>>B = A(1:2:5, :)
 B  =

    1.     2.     3.
    7.     8.     9.
    13.    14.    15.
```

Nesse exemplo, retornou apenas as linhas ímpares com todas as colunas.

E alterando elementos da matriz:

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    13.    14.    15.

>>A(1, :) = 0 % altera da linha 1, todos os
elementos das colunas para 0
 A  =

    0.     0.     0.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    13.    14.    15.
```

```
>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    13.    14.    15.

>>A(1:2:5, :) = 0 % Altera as linhas 1, 3 e 5 de
todas as colunas com o valor 0
 A  =

    0.     0.     0.
    4.     5.     6.
    0.     0.     0.
    10.    11.    12.
    0.     0.     0.

>>A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]
 A  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.
    13.    14.    15.

>>A(1:2:5, 1:2) = 0 % Altera as linhas 1, 3 e 5,
colunas 1 e 2 com o valor 0
 A  =

    0.     0.     3.
    4.     5.     6.
    0.     0.     9.
    10.    11.    12.
    0.     0.     15.
```

6.4.3. Localizando a posição de elementos (comando FIND)

O comando FIND é utilizado para encontrar a posição (ou posição) dos elementos que coincidem com a pesquisa.

Por exemplo, considere o vetor:

```
>> X = [1 0 4 –3 0 0 0 8 6];
```

E o comando

```
>> indices = find(X == 0)
indices =

2   5   6   7
```

O comando localizou todos os valores iguais a 0 (X == 0) e retornou as posições desses elementos.

O mesmo serve se o vetor estiver em forma de coluna:

```
>> X = [1; 0; 4; –3; 0; 0; 0; 8; 6];

>> indices = find(X == 0)
indices =

2
5
6
7
```

6.5. FIND EM MATRIZES.

Para utilizer o comando FIND em matrizes, há algumas diferenças:

```
>> X = [1 0 4;
        –3 0 0;
        0 8 6]
X =

 1   0   4
–3   0   0
 0   8   6

>> indices = find(X == 0)
indices =

3
4
5
8
```

Nesse exemplo, o comando FIND retornou a posição linear dos valores iguais a 0. A posição linear é definida pela colunas, começando no canto superior esquerdo:

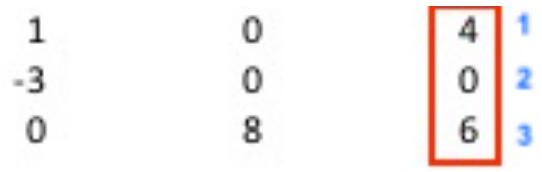
Os números em Azul representam a posição dos elementos.

6.5.1. Localizando elementos em colunas/linhas de matrizes

Podemos ainda procurar um elemento numa coluna ou linha especifica, nesse caso, o comando FIND irá retornar o(s) elemento(s) na posição referente a coluna ou linha:

```
>> indices = find(X(:, 3) == 0)
indices =  2
```

Nesse exemplo, o comando FIND irá procurar em todas as linhas da coluna 3 pelo valor igual a 0.



Retornando a posição 2.

6.5.2. Utilizando o valor retornado pelo FIND

Aproveitando o exemplo anterior, em que localizamos o valor 0 na 3ª coluna:

```
>> indices = find(X(:, 3) == 0)
indices =  2
```

Percebemos que há o valor 0 na 2ª linha da referida coluna, com isso vamos utilizar esse valor para retornar a linha toda e armazenar na variável Res:

```
>> Res = X(indices, :)
Res =
```

```
−3    0    0
```

Graficamente falando:

6.5.3. Exemplos utilizando o comando FIND

Veja outros exemplos com o uso do comando FIND:

```
>> indices = find(X(:, 2) == 0)
indices =

1
2
```

Nesse caso, o comando FIND retornou duas ocorrência de valores iguais a 0, e no momento de selecionar as linhas, que são utilizados para apresentar o resultado:

```
>> Res = X(indices, :)
Res =

1   0   4
−3   0   0
```

Ou podemos ainda querer que ele retorne apenas a 1ª ocorrência do índice:

```
>> Res = X(indices(1), :)
Res =

1   0   4
```

Compare os dois casos para entender o que foi alterado.

6.6. OPERADORES ESPECIAS PARA INICIALIZAÇÃO DE VETORES E MATRIZES

No Octave há comandos especiais para definir valores de vetores e matrizes:

eye	Cria uma matriz identidade
linspace	Criar um vetor linearmente espaçado
ones	Cria um vetor ou matriz com uns
zeros	Cria um vetor ou matriz com zeros
rand	gera uma matriz com valores aleatórios

6.6.1. Função eye()

A função eye() cria uma matriz identidade, que é uma matriz em que os elementos da diagonal principal são iguais a 1, os outros elementos são iguais a 0.

Exemplos:

```
>>id1 = eye(4, 4)
 id1  =

    1.    0.    0.    0.
    0.    1.    0.    0.
    0.    0.    1.    0.
    0.    0.    0.    1.

>>id2 = eye(4, 2)
 id2  =

    1.    0.
    0.    1.
    0.    0.
    0.    0.
>>a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
 a  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.

>>id = eye(a)
 id  =

    1.    0.    0.
    0.    1.    0.
    0.    0.    1.
    0.    0.    0.
```

A sintaxe do comando eye():

```
X = eye(m,n)
X = eye(A)
```

Onde:

Onde m e n são valores que representam o número de linhas e colunas.

A: uma outra forma de construir uma matriz identidade, passa-se uma matriz A e o comando eye() constrói uma matriz identidade com as mesmas dimensões da matriz A.

6.6.1. Função linspace()

Cria um vetor vetorial espaçado linearmente.

```
>>linspace(1,2,11)
 ans  =

    1.    1.1    1.2    1.3    1.4    1.5    1.6
1.7    1.8    1.9    2.

>>linspace(0,9,10)
 ans  =

    0.    1.    2.    3.    4.    5.    6.    7.
8.    9.

>>linspace(0, 10, 5)
 ans  =

    0.    2.5    5.    7.5    10.
```

6.6.2. Função ones

Gera uma matriz onde todos os elementos são iguais a 1

```
>>m1 = ones(1, 4)
 m1  =

    1.    1.    1.    1.

>>m2 = ones(3, 2)
 m2  =
```

```
    1.    1.
    1.    1.
    1.    1.

>>a = [1 2 3; 4 5 6]
 a  =

    1.    2.    3.
    4.    5.    6.

>>m3 = ones(a)
 m3  =

    1.    1.    1.
    1.    1.    1.
```

As formas de chamar a função one são:

```
y=ones(m1,m2,...)
y=ones(A)
y=ones()
```

Onde:

m1 e m2 são valores que representam o número de elementos em cada dimensão

A: uma outra forma de construir uma matriz identidade, passa-se uma matriz A e o comando ones() constrói uma matriz de elementos 1 com as mesmas dimensões da matriz A.

6.6.3. Função zeros

Semelhante a função ones, preenche uma matriz com os elementos iguais a 0 (zero):

```
>>m1 = zeros(1, 5)
 m1  =

    0.    0.    0.    0.    0.

>>m2 = zeros(3, 4)
 m2  =

    0.    0.    0.    0.
    0.    0.    0.    0.
```

```
    0.    0.    0.    0.

>>a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
 a  =

    1.     2.     3.
    4.     5.     6.
    7.     8.     9.
    10.    11.    12.

>>m = zeros(a)
 m  =

    0.    0.    0.
    0.    0.    0.
    0.    0.    0.
    0.    0.    0.
```

A sintaxe da função zeros é:

```
y=zeros(m1,m2,..)
y=zeros(A)
y=zeros()
```

Onde:

m1 e m2 são valores que representam o número de elementos em cada dimensão

A: uma outra forma de construir uma matriz identidade, passa-se uma matriz A e o comando zeros() constrói uma matriz de elementos 1 com as mesmas dimensões da matriz A.

6.6.4. Função rand

A função rand é utilizada para criar números e matriz aleatórias

```
>>x=rand(3,3)
 x  =

    0.5667211    0.0568928    0.7279222
    0.5711639    0.5595937    0.2677766
    0.8160110    0.1249340    0.5465335


>>a = [1 2 3; 4 5 6]
```

```
a  =

   1.    2.    3.
   4.    5.    6.
```

A sintaxe de chamar a função rand:

```
rand(m1,m2,.. [,key])
rand(x [, key])
rand()

rand(key)
rand("seed" [,n])
rand("info")
```

Onde**:**

Sem argumento "key", as sintaxes abaixo produzem matrizes randômicas com o gerador randômico corrente.

rand(m1,m2) é uma matriz randômica de dimensão m1 por m2.

rand(m1,m2,..,mn) é uma matriz randômica de dimensão m1 por m2,.. por mn.

rand() : sem argumentos, fornece um escalar cujo valor muda a cada vez que é referenciado.

6.7. EXERCÍCIOS

1. Criação e operação de matrizes
    a. Criar 2 matrizes A e B, com dimensões 3X4.
    b. Criar em C a transporta de A.
    c. Gerar D, somando A e B.
    d. Gerar E, subtraindo B de A .
    e. Gerar F com a multiplicação, elemento a elemento, de A por B.
    f. Adicionar 5 a cada elemento de B.
    g. Apagar todas as variáveis usadas até este momento.

2. Operação em vetores:
    a. Gerar um vetor inteiro H, com o valor inicial 6 e valor máximo 100, com variação entre elementos de 6.
    b. Gerar um vetor I, subtraindo 2 de cada um dos valores de H.
    c. Gerar um vetor J com a multiplicação, elemento a elemento, de H por I.
3. Insira a matriz X = [2 7 9 7; 3 1 5 6; 8 4 2 5] no Octave, execute os seguintes comandos e detalhe o valor obtido
    a. A = X(1, 1:3)
    b. B = X(1:2, 1:2)
    c. C = X(:,2:2:$) % O operador $ é utilizado para designar o fim da série
    d. D = X(1:2:$, :)
4. Detalhe o que cada comando irá fazer:
    a. x = [3 15 9 12 -1 0 -12 9 6 1]
       x (x > 0) = 0
    b. x = [3 15 9 12 -1 0 -12 9 6 1]
       x(:, 1:2:$) = 2
    c. x = [3 15 9 12 -1 0 -12 9 6 1]
       y = x(x>10)
    d. x = [3 15 9 12 -1 0 -12 9 6 1]
       y = x(modulo(x,2)==0)
5. Faça:
    a. Criar com a função rand a matriz real **mat**, de dimensões 7X5 e multiplicá-la por 100. Depois utilizando a função **int**, modifique a matriz **mat** de forma que contenha apenas valores inteiros
    b. Com a matriz **mat** obtida do item anterior apresente a soma e o produto de todos seus valores;
    c. Ainda sobre a matriz mat apresente o maior valor, o menor valor e a média dos valores

7. SALVANDO E RECUPERANDO DADOS EM ARQUIVO

O Octave possui dois comandos básicos: *save* e *load* que permite salvar e ler dados de arquivos em vários formatos. Vale lembrar que esses comandos salvam apenas os dados, os comandos devem ser salvos em arquivos a parte.

7.1. SALVANDO DADOS EM ARQUIVO

A sintaxe do comando *save* é a seguinte:

```
save [options] file [variables]
```

Onde:

Options: são as opções aceita pelo comando, as principais são:

-append	Adiciona o novo conteúdo ao arquivo, sem destruir os dados anteriores
-ascii	Salva os dados num arquivo texto simples, sem informações a mais
-binary	Salva os dados em formato binário (o arquivo fica menor, mas de difícil entendimento se aberto por um editor de texto)
-text	Salva os dados em formato texto (padrão)
-zip ou -z	comprime os dados com o algoritimo gzip

File: arquivo onde os dados serão salvos.

Variables: pode ser uma ou mais variáveis que terão seu conteúdo salvo no arquivo.

7.1.1. Exemplo 1

Por exemplo, considere os seguintes comandos:

```
>>a = [1 2 4; 6 8 4; 1 4 5];

>>save dados.txt a
```

Irá salvar o arquivo “dados.txt” com o seguinte conteúdo:

```
# Created by Octave 4.0.3, Mon Jan 22 14:20:37 2018 -02 <cendron@Marcelos-MacBook-Pro.local>
# name: a
# type: matrix
# rows: 3
# columns: 3
1 2 4
6 8 4
1 4 5
```

7.1.2. Exemplo 2

Se forem enviados três variáveis para o arquivo:

```
>>a = [1 2 4; 6 8 4; 1 4 5];
>>b = 3.444;
>>c = "texto";

>>save dados.txt a b c
```

O arquivo resultante fica:

```
# Created by Octave 4.0.3, Mon Jan 22 14:29:30 2018 -02 <cendron@Marcelos-MacBook-Pro.local>
# name: a
# type: matrix
# rows: 3
# columns: 3
1 2 4
6 8 4
1 4 5


# name: b
# type: scalar
3.444

# name: c
# type: string
# elements: 1
# length: 5
texto
```

7.2. RECUPERANDO DADOS DE ARQUIVOS

Quando se quer ler os dados de arquivo, utilizamos o comando Load, que tem a seguinte sintaxe:

```
load [options] file [variables]
```

`options`: As opções são parecidas as dos comando *save*:

-ascii	Assume que os dados estão em formato de texto sem nenhum cabeçalho a mais.
-binary	Assume que o arquivo está em formato binário
-text	Assume que o arquivo está em formato texto

`File`: nome do arquivo a ser lido

`Variables:` As variáveis que serão armazenadas

7.2.1. Exemplo 1

Para exemplificar o uso do Load, vamos partir do arquivo "dados.txt" criado com os seguintes dados:

```
# name: AA
# type: matrix
# rows: 3
# columns: 3

4 5 6
1 2 3
8 9 10
```

Como pode ser observado, os números estão organizados numa matriz de 3 linhas por 3 colunas, do tipo matriz, com o nome "AA"

Quando executamos o comando:

```
>>load dados.txt
```

Será criado uma variável com o nome "AA" e os dados que estão no arquivo:

```
>> AA
```

```
AA =

4     5     6
1     2     3
8     9    10
```

7.2.2. Exemplo 2

Outro exemplo, dessa vez, o arquivo tem os seguintes dados:

```
# name: AA
# type: matrix
# rows: 3
# columns: 3

4 5 6
1 2 3
8 9 10

# name: BB
# type: matrix
# rows: 2
# columns: 3

11 12 13
14 15 16
```

Executando o comando:

```
>>load dados.txt
```

Obteremos duas variáveis: AA e BB com os valores:

```
>> AA

 AA =
      4     5     6
      1     2     3
      8     9    10

>> BB

BB =
      11    12    13
```

```
    14    15    16
```

7.2.3. Exemplo 3

Dessa vez, iremos querer carregar os dados de um arquivo sem a estrutura de cabeçalho, apenas os dados:

```
4 5 6
1 2 3
8 9 10
```

Nesse caso, mudamos a forma de entrada do comando load:

```
>> a = load("dados.txt")
     a =

     4     5     6
     1     2     3
     8     9    10
```

Veja que antes do comando load foi especificada a variável em que os dados serão salvos. Caso seja o omitido o nome da variável, será criada uma com o mesmo nome do arquivo.

7.2.4. Exemplo 4

Caso se imagine que para salvar duas matrizes basta colocar uma abaixo da outra:

```
4 5 6
1 2 3
8 9 10

1 2 3
11 12 13
```

Ao tentar carregar os dados, vamos apenas 1 matriz com todos os valores:

```
>> a = load("arquivo.txt")
a =
```

```
 4     5     6
 1     2     3
 8     9    10
 1     2     3
11    12    13
```

Para recuperar duas matrizes separadas é necessário especificar os cabeçalhos para cada uma, como feito no exemplo 2.

7.3. EXERCÍCIOS

1. Dado a lista de valores abaixo, crie um arquivo com esses valores e faça a importação no Octave:

```
2
5
9
10
11
15
16
17
18
20
```

2. Faça um programa que leia 2 valores de um arquivo e verifique qual o número é maior
3. Faça um programa que receba do teclado 2 valores e salve os dois números num arquivo.

8. GRÁFICOS

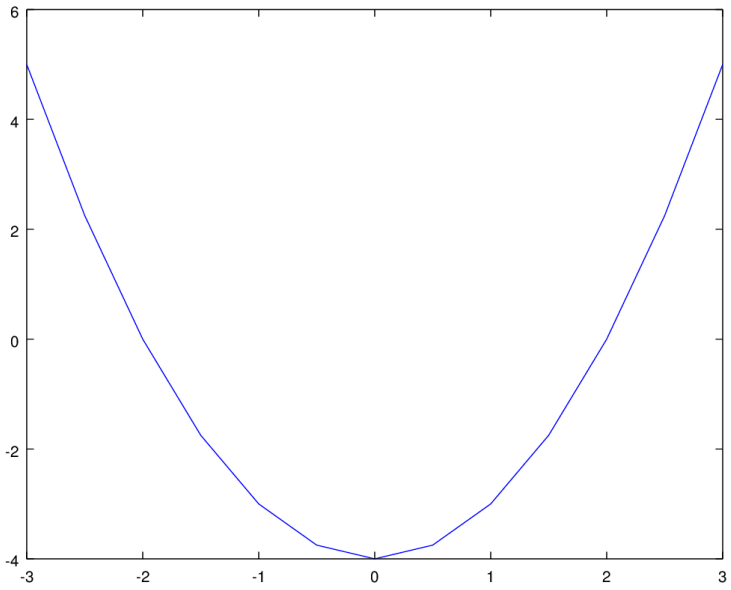
O Octave permite criar diversos tipos de gráficos em 2 e 3 dimensões, para isso, a base de gráficos é construída através da função **plot**.

8.1. GRÁFICOS EM 2 DIMENSÕES

Para criar um gráfico 2D, é utilizado a função **plot**, que necessita de pelo menos 2 parâmetros, um com a lista de valores do eixo X e outro com os valores de Y:

```
>>X = [-3:0.5:3]
 X  =

  - 3.  - 2.5  - 2.  - 1.5  - 1.  - 0.5    0.    0.5
1.    1.5    2.    2.5    3.

>>Y = X.^2.-4
 Y  =

   5.    2.25    0.  - 1.75  - 3.  - 3.75  - 4.  - 3.75
- 3.  - 1.75    0.    2.25    5.

>>plot(X, Y)
```

E o resultado:



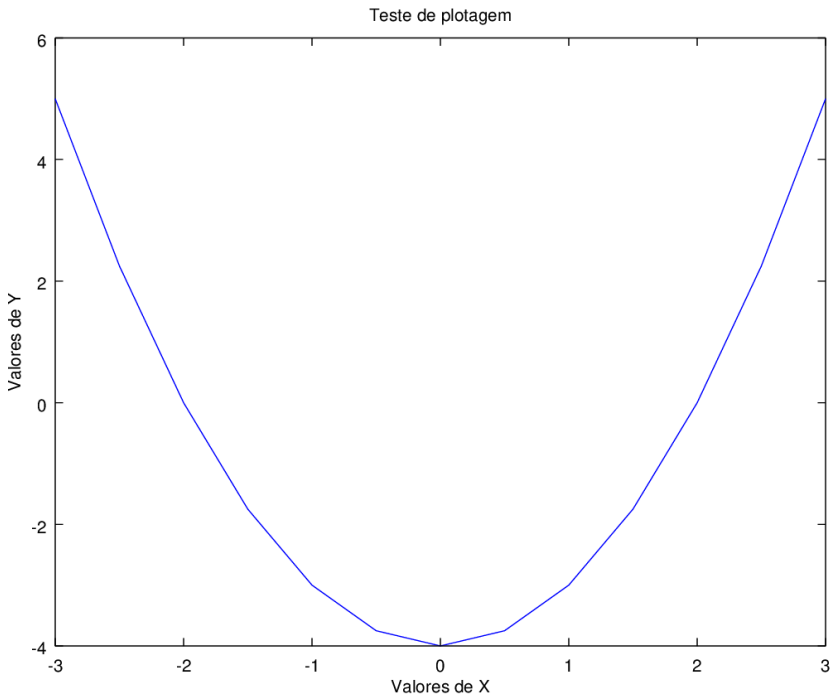
8.1.1. Inserindo título e ajustando eixos

O Octave permite que se crie títulos e adicione rótulos nos eixos dos gráficos. Os comandos:

- **title():** adiciona um título na parte superior do gráfico;
- **xlabel():** adiciona um rótulo no eixo das abscissas;
- **ylabel():** adiciona um rótulo no eixo das ordenadas.

Adicionando esses campos ao exemplo anterior:

```
>>X = [-3:0.5:3]
 X  =

  - 3.  - 2.5  - 2.  - 1.5  - 1.  - 0.5    0.    0.5
1.    1.5    2.    2.5    3.

>>Y = X.^2.-4
 Y  =

   5.    2.25    0.  - 1.75  - 3.  - 3.75  - 4.  - 3.75
- 3.  - 1.75    0.    2.25    5.

>>plot(X, Y);

>>xlabel("Valores de X");

>>ylabel("Valores de Y");

>>title("Teste de plotagem");
```

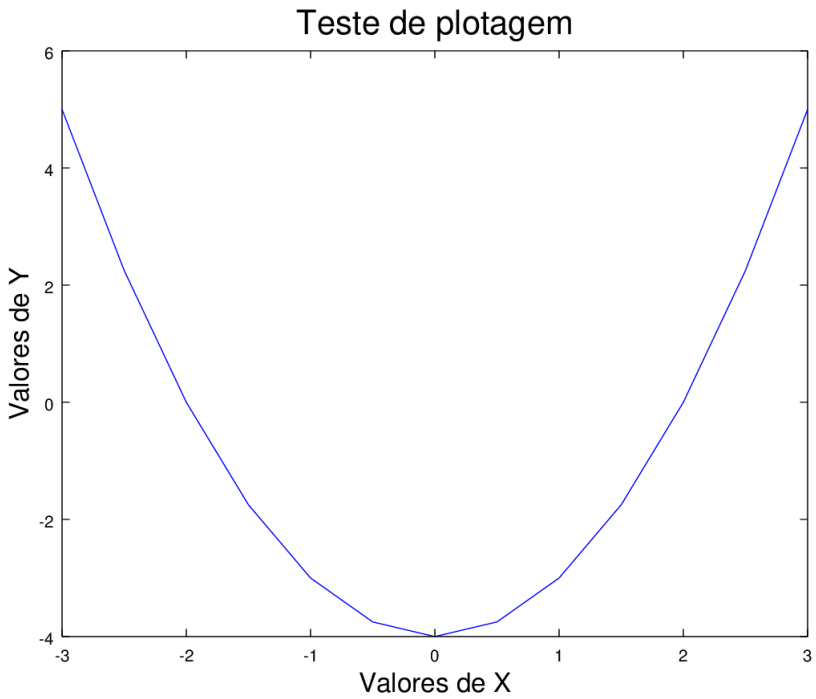


Podemos personalizar melhor o gráfico passando certos parâmetros para as funções **xlabel**, **ylabel** e **title:**

Por exemplo:

```
>>xlabel("Valores de X", "fontsize", 16);

>>ylabel("Valores de Y", "fontsize", 16);

>>title("Teste de plotagem","fontsize", 20);
```

Irá reproduzir o seguinte gráfico:



Algumas propriedades que podem ser alteradas são:

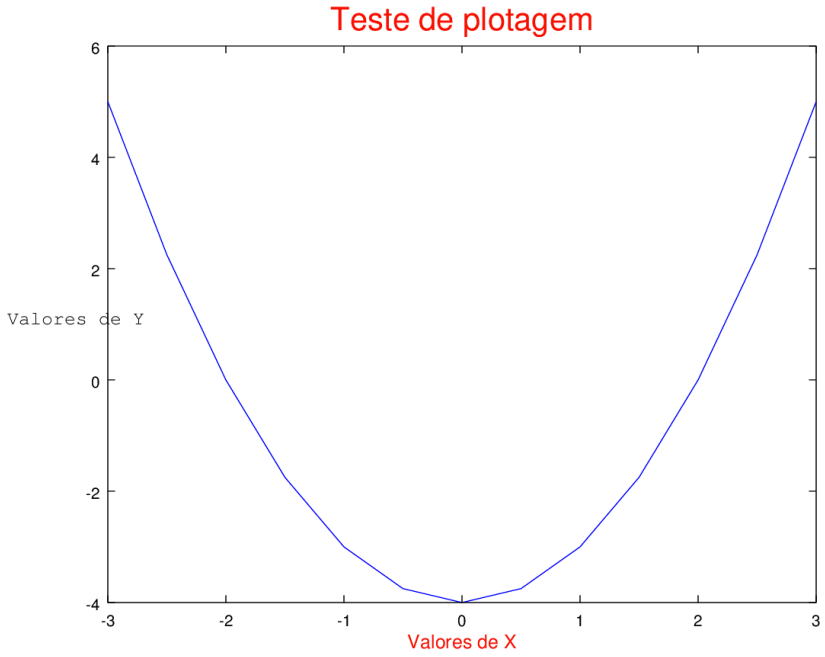
- “fontname”: Define a fonte, o valor passado deve ser uma string
- “fontsize”: Define o tamanho da fonte. O comando espera um valor numérico, padrão 10. A unidade de medida é definida pelo comando `fontunits`
- “fontunits”: Define a unidade de medida da fonte, aceita umas das opções "centimeters", "inches","normalized", "pixels", "points". O padrão é "points".
- “fontweight”: espessura da fonte, é aceita umas das opções: "bold", "demi", "light", "normal", o padrão é "normal".
- “horizontalalignment”: define o alinhamento horizontal, os valores aceitos são: "center", "left", "right", o padrão é "left".
- “verticalalignment”, definie o alinhamento vertical, valores aceitos: "baseline", "bottom", "cap", "middle”, "top". O padrão é "middle”.
- “rotation”: Define a rotação do texto, o valor aceito deve ser no formato numérico e é medido em graus
- “color”: A cor do texto, o valor aceito é uma cor em inglês no formato string

Esse parâmetros podem ser passados na mesma linha de configuração.

Veja o exemplo:

```
>>X = [-3:0.5:3];
>>Y = X.^2.-4;

>>plot(X, Y);

>>xlabel("Valores de X", "fontsize", 12, "color", "red",
"verticalalignment", "top" );
>>ylabel("Valores de Y", "fontsize", 12, "fontname",
"Courier", "fontweight", "demi", "rotation", 0 );
>>title("Teste de plotagem","fontsize", 20, "color",
"red");
```

E o resultado:



8.1.2. Adicionando legenda

Para adicionar legendas usa-se o comando:

```
Legend( strings [,"location",pos][,"orientation", orient
][, opções ] )
```

onde:

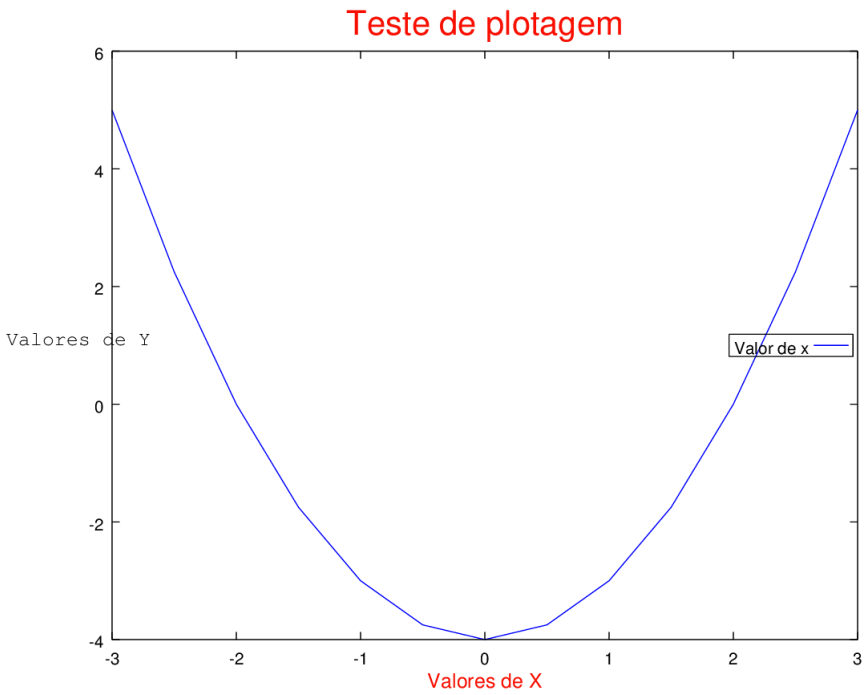
- strings: os textos que irão aparecer na legenda
- "location": esse comando é opcional, define a posição da legenda, caso seja utilizado, o argumento `pos` deve ser um desses:

pos	Localização da legenda
north	Topo centro
south	Baixo centro
east	Centro direita
west	Centro esquerda
northeast	Topo direita(padrão)
northwest	Topo Esquerda
southeast	Baixo direita
southwest	Baixo esquerda
outside	Pode ser adicionado ao final da posição, para colocar a caixa da legenda fora da área do gráfico. Por exemplo “**eastoutside**”

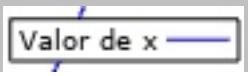
- “orientation”: comando opcional, define a orientadação da legenda, se for utilizado, o valor de `orient` deve ser "vertical" (padrão) ou "horizontal".

  Veja um exemplo:

```
>>legend("Valor de x", "location", "east", "orientation", "vertical");
```



Ainda o comando Legend pode ser executado, apenas com um dos argumentos:

"show"	Mostra a legenda no plot
"hide"	Oculta a legenda no plot
"toggle"	Altera o modo entre "hide" e "show" (se estiver oculto, exibe o plot e se estiver exibindo, oculta)
"boxon"	Mostra uma caixa ao redor da legenda (padrão) 
"boxoff"	Oculta uma caixa ao redor da legenda. 
"right"	Posiciona o rótulo de texo no lado direito do figura (padrão) 
"left"	Posiciona o rótulo de texo no lado esquerdo do figura 
"off"	Apaga a legenda.

8.1.3. Outros parâmetros do gráfico

Alguns outros comandos interessante para a configuração do gráfico:

8.1.3.1. Mudando a cor do fundo

A cor de fundo da janela gráfica do Octave é branca por padrão. É possível mudá-la através do comando:

```
set( gca() , "color" , cor )
```

Onde:

gca() retorna como parâmetro o manipulador de eixos da janela gráfica corrente,

O parâmetro **cor** deve ser uma cor em inglês

8.1.3.2. Linhas de grade

As linhas de grade pode ser adicionadas através do comando:

```
grid
```

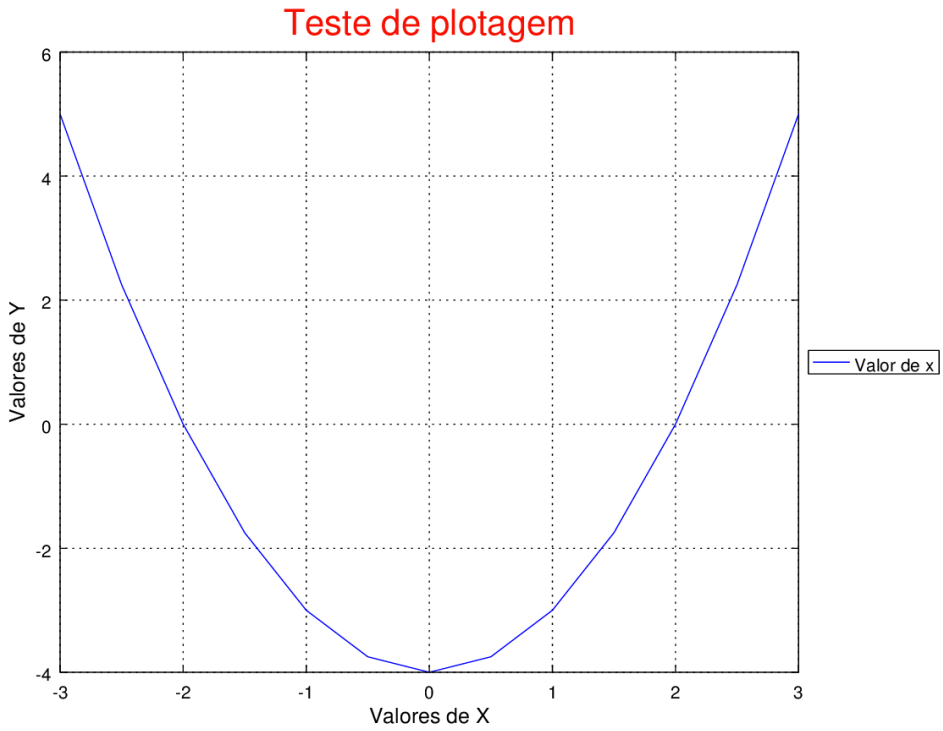
com a opções:

- on
- off
- minor
- minor on
- minor off

Por exemplo:

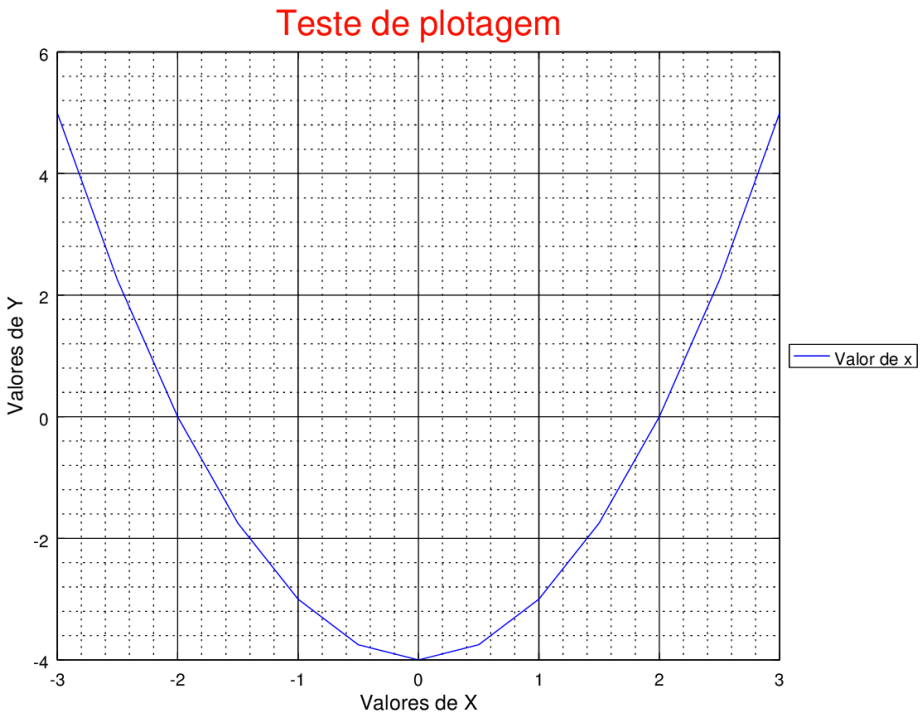
```
>>grid on
```

Irá produzir o seguinte gráfico:



Pode-se ainda inserir o grid menor:

```
>>grid minor on
```

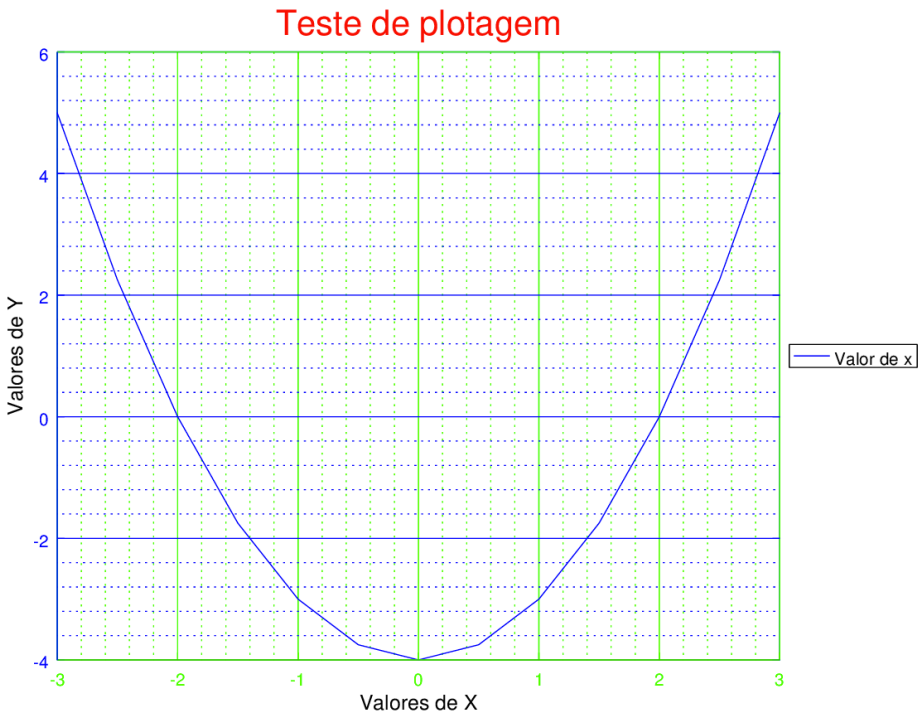


Para ajustar a cor do grid, precisamos especificar a cor através do comando:

```
>> set(gca(), "xcolor", cor)
>> set(gca(), "ycolor", cor)
```

Por exemplo, para setar a cor verde para o eixo X e azul para o Y:

```
>> set(gca(), "xcolor", "green")
>> set(gca(), "ycolor", "blue")
```


Teste de plotagem
6
4
2
0
-2
-4
-3
-2
-1
0
1
2
3
Valores de Y
Valores de X
Valor de x

9. COMANDOS DE REPETIÇÃO

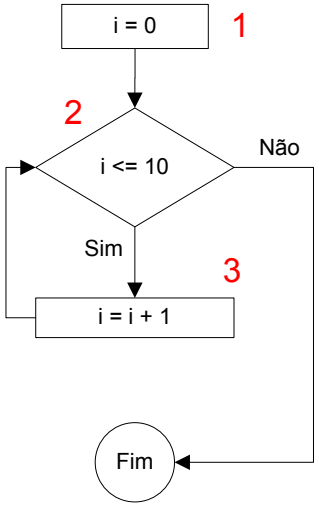
Utilizamos os comandos de repetição quando desejamos que um determinado conjunto de instruções ou comandos sejam executados um número definido ou indefinido de vezes, ou enquanto um determinado estado prevalecer ou até que seja alcançado.

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 apresentas as três partes:




```
i = 0
 while i < 10
    i = i + 1
end
```

Figura 17 - Partes de um laço de repetição

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

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

9.1. 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:

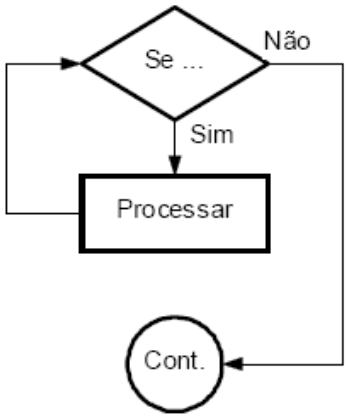
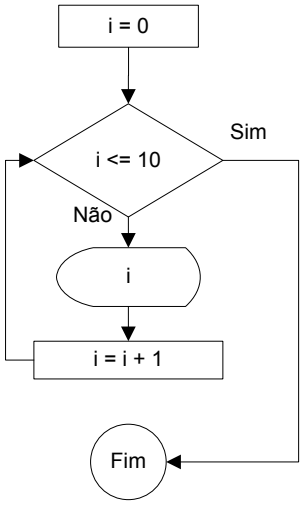


Figura 18 - Diagrama While

Exemplo de contador:




```
i = 0;
while i < 10
   i = i + 1;
   printf("%d\n", i);
end
```

Figura 19 - Laço While

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

A construção do Laço **For**, em diagrama é muito semelhante com o laço **While**, porém o uso dele é mais abrangente e flexível, veja alguns exemplos:

```
for i=1:10
   printf("%d\n", i)
end
```

Outro exemplo, dessa vez, o for irá interagir numa lista:

```
a = [1,3,2,4];
for i=a
    disp(i);
end
```

Nesse exemplo, a variável a é uma lista com 4 elementos. O *for* interage sobre cada objeto, imprimindo um de cada vez.

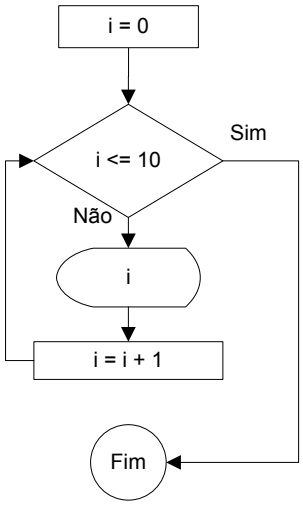
9.3. CASOS PRÁTICOS COM REPETIÇÃO

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

9.3.1. Exemplo 01 - Somando os valores

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

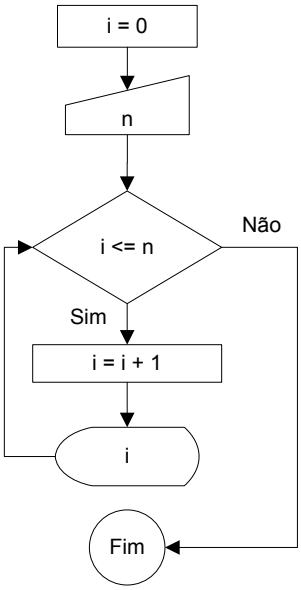
Partimos do laço de repetição já apresentando *While*:




```
i = 0;
while i < 10
    i = i + 1;
end
```

Figura 20 - 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.




```
i = 0;
n = input("Digite o maior valor:");
while i <= n
    i = i + 1;
    printf("%d\n", i)
end
```

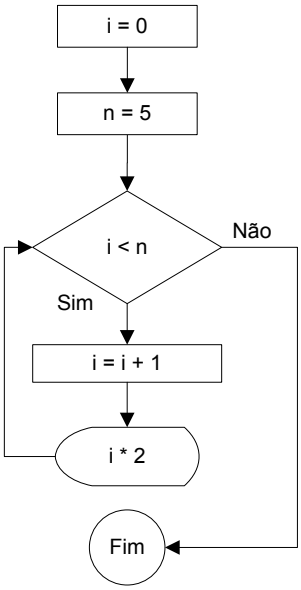
Figura 21 - Modificações no laço básico

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:

```
Total = 2 + 4 + 6 + 8 + 10
Total = 30
```

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:

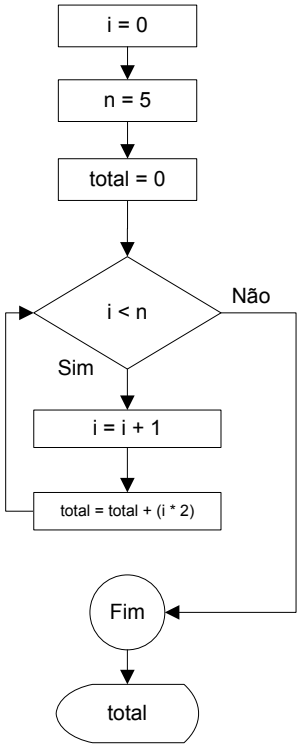



```
i = 0;
n = input("Digite o maior valor:");
while i < n
    i = i + 1;
    valor = i * 2;
    printf("%d\n", valor);
end
```

Figura 22 - 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:




```
i = 0;
n = input("Digite o maior valor:");
total = 0;
while i < n
    i = i + 1;
    total = total + (i * 2)

end

printf("Valor final: %d\n", total)
```

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**
1°	0	1	0 + (1 * 2) = 2
2°	2	2	2 + (2 * 2) = 6
3°	6	3	6 + (3 * 2) = 12
4°	12	4	12 + (4 * 2) = 20
5°	20	5	20 + (5 * 2) = 30
Saída	30		

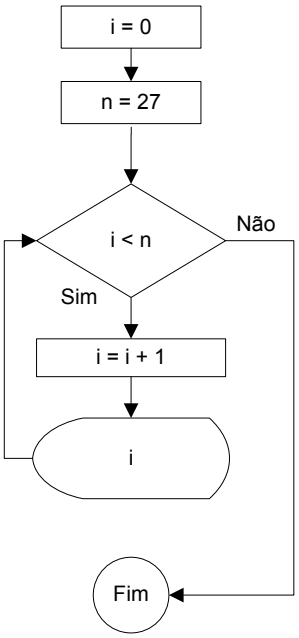
Matematicamente falando a fórmula do nosso exemplo é:

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

9.3.2. Exemplo 02 – Verificando os divisores.

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

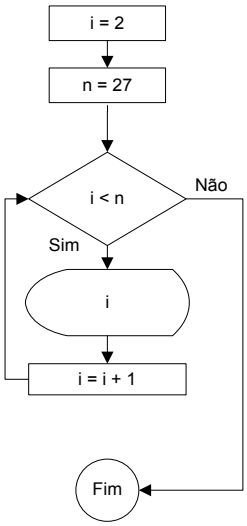
Para facilitar a resolução vamos partir utilizando o laço de repetição *While*:




```
i = 0;
n = 27;
total = 0;
while i < n
    i = i + 1;
    printf("%d\n", i)
end
```

Figura 23 - 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:




```
i = 2;
n = 27;
total = 0;
while i < n
    printf("%d\n", i)
    i = i + 1;
end
```

Figura 24 - Modificações no laço While

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

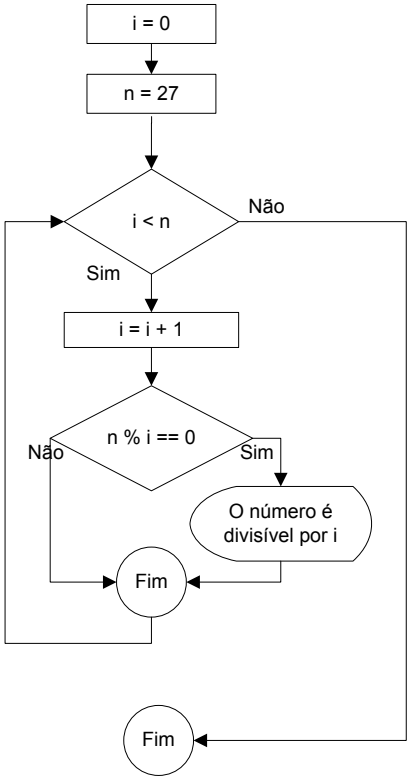


Figura 25 - Diagrama do programa

```
i = 2;
n = 27;
total = 0;
while i < n
    if mod(n, i) == 0
        printf("O número %d é divisível por %i\n", n, i);
    end
    i = i + 1;
end
```

Figura 26 - Código-fonte

9.4. 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.