lista (2)

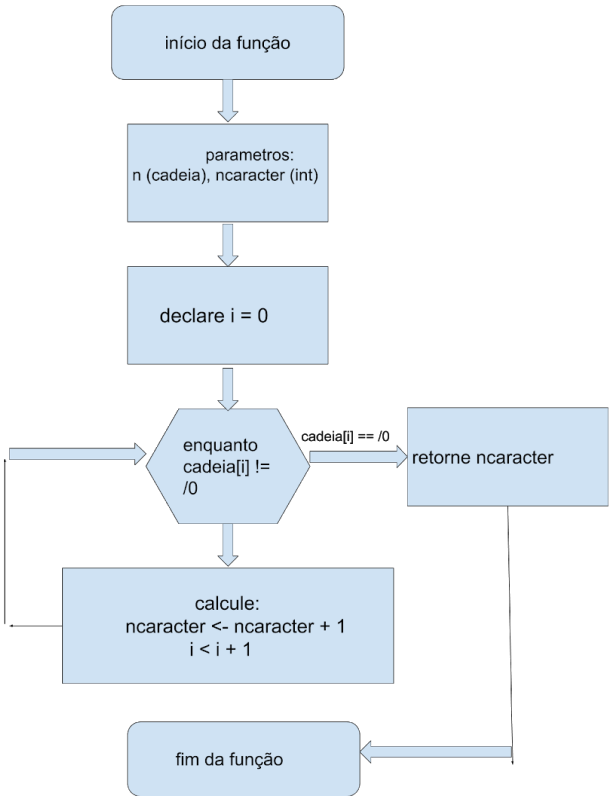
exercício 1)

1- início
2- função comprimento
3- parâmetros: n tipo cadeia, ncaracter tipo int
5- enquanto n != ‘/0’ faça
    5.1- ncaracter <- ncaracter + 1
6- fim enquanto
7- retorna cadeia
8- fim

linguagem natural

1- início
2- declarar a função
3- declarar um parâmetro tipo cadeia e o outro para a quantidade de caracteres
4- enquanto número for diferente de ‘/0’ somar 1 na quantidade de caracteres
5- retornar cadeia
6- fim função

fluxograma



teste de mesa

		variáveis	
		número caracter	cadeia
5.1	- ncaracter <- ncaracter + 1	1	[a,b,c,/o]
5.1	- ncaracter <- ncaracter + 1	2	[a,b,c,/o]
5.1	- ncaracter <- ncaracter + 1	3	[a,b,c,/o]

exercício 2)

1- função Évazio
2- parâmetro: n tipo cadeia, vazio tipo booleano
3- se n[0] == “”
    3.1 declare- vazio = verdadeiro
4- se não
    4.1 declare- vazio = falso
5- fim se
6- retorne vazio
7- fim évazio

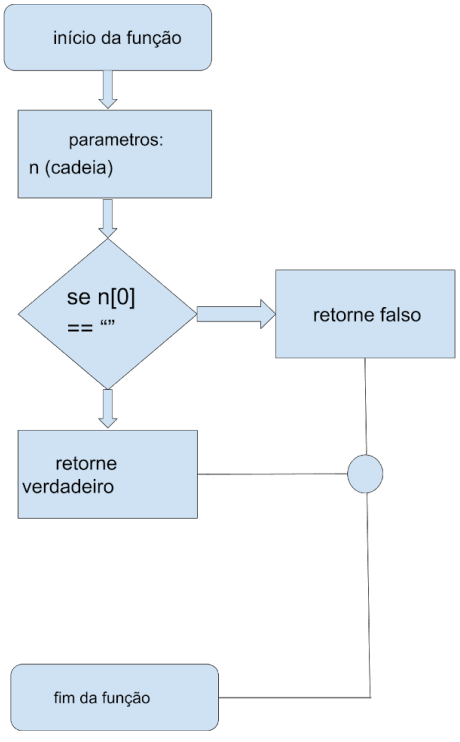
linguagem natural

1- início função
2- declarar parâmetro para cadeia e outro para receber o booleano
3- se cadeia for vazia coloque o valor de vazio verdadeiro
4- se não coloque o valor de vazio falso
5- retorne valor de vazio
6- fim da função

teste de mesa

passo		cadeia	vazio
3	Se cadeia[0] == “”	[]	
3.1	vazio = verdadeiro	[]	
			verdadeiro
3	Se cadeia[0] == “”	[a, b]	
4.1	vazio = falso	[a, b]	falso

fluxograma

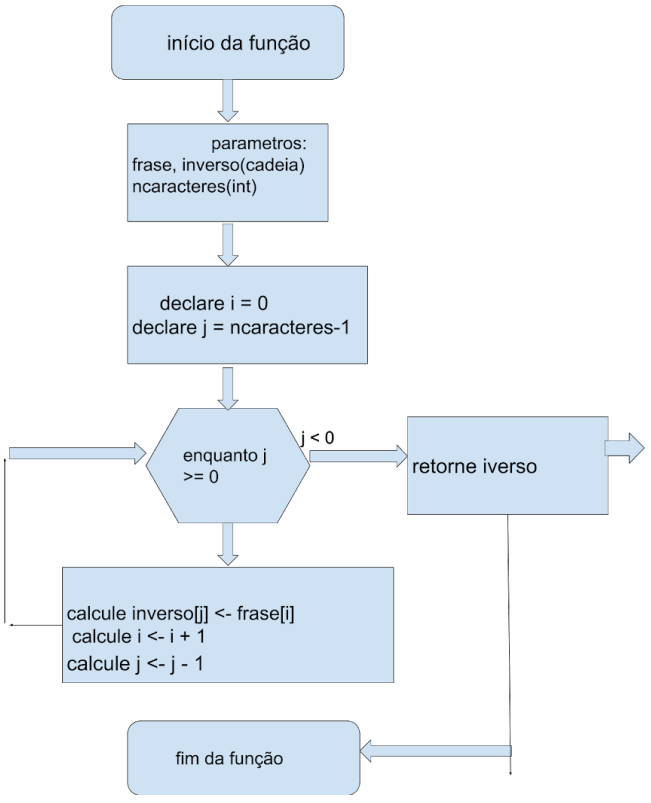


exercício

4)
1- função inversor
2- parâmetro: frase , inverso, tipo cadeia
3- declare ncaracteres
4- declare i = 0
5- declare j = ncaracteres - 1
6- enquanto j >= 0 faça
    6.1- calcule frase[i] <- inverso[j]
    6.2- calcule i <- i + 1
    6.3- calcule j <- j - 1
7- retorne inverso
8- fim inversor

linguagem natural
1- declarar a função
2- declarar os parâmetros tipo cadeia para receber a frase e o inverso dela
3- declarar uma variável para a quantidade de caracteres
4- declare um contador para a frase normal valendo 0
5- declare outro contador para o inverso da frase valendo -1
6- enquanto o contador para o inverso for menor ou igual a zero
7- modifique o primeiro da frase pelo último do inverso usando os contadores
8- calcule + 1 nos contadores a cada inversão
9- retorne o inverso
10- fim função

fluxograma



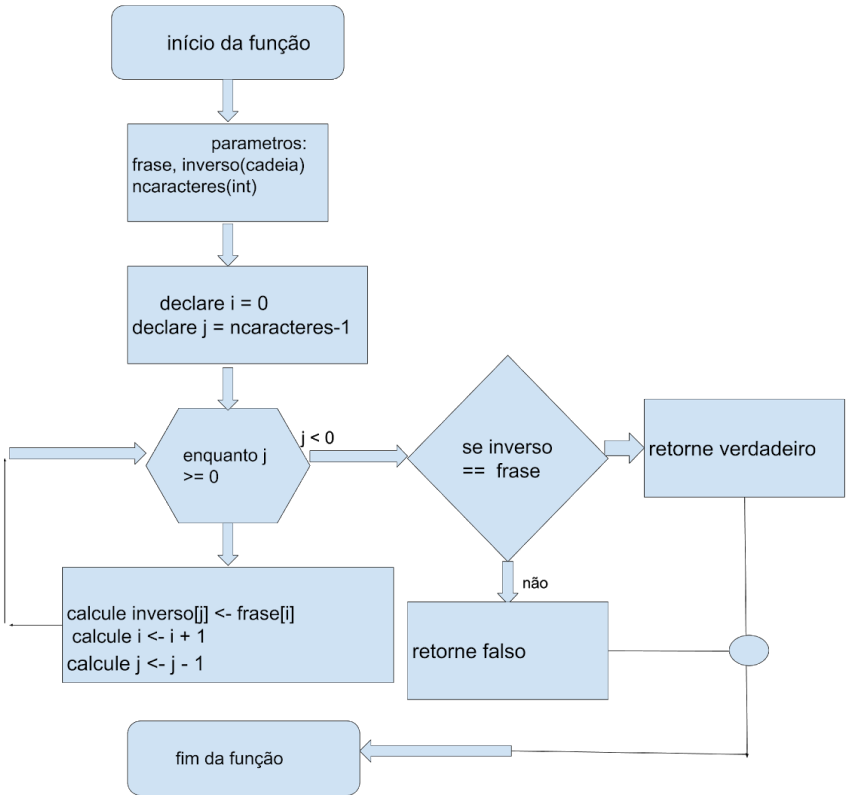
teste de mesa

passo	comando	frase	inverso	ncaracteres	i	j
3	i = 0	[o, l, a]	[]	3	0	
4	j = ncaracteres- 1	[o, l, a]	[]	3	0	2
5.1	frase[i] <- inverso[j]	[o, l, a]	[a]	3	0	2
5.2	i <- i + 1	[o, l, a]	[a]	3	1	2
5.3	j <- j - 1	[o, l, a]	[a]	3	1	1
5.1	frase[i] <- inverso[j]	[o, l, a]	[a, l]	3	1	1
5.2	i <- i + 1	[o, l, a]	[a, l]	3	2	1
5.3	j <- j - 1	[o, l, a]	[a, l]	3	2	0
5.1	frase[i] <- inverso[j]	[o, l, a]	[a, l, o]	3	2	0
5.2	i <- i + 1	[o, l, a]	[a, l, o]	3	3	0
5.3	j <- j - 1	[o, l, a]	[a, l, o]	3	3	-1

5-
1- função palíndromo
2- parâmetro: frase , inverso, tipo cadeia
3- declare ncaracteres
4- declare i = 0
5- declare j = ncaracteres - 1
6- enquanto j >= 0 faça
    6.1- calcule frase[i] <- inverso[j]
    6.2- calcule i <- i + 1
    6.3- calcule j <- j - 1
7- se frase = inverso
    7.1- mostrar verdadeiro
8- se não mostrar falso
9- fim palíndromo

linguagem natural
1- declarar a função
2- declarar os parâmetros tipo cadeia para receber a frase e o inverso dela
3- declarar uma variável para a quantidade de caracteres
4- declare um contador para a frase normal valendo 0
5- declare outro contador para o inverso da frase valendo -1
6- enquanto o contador para o inverso for menor ou igual a zero
7- modifique o primeiro da frase pelo último do inverso usando os contadores
8- calcule + 1 nos contadores a cada inversão
9- se frase for igual ao inverso retorne verdadeiro
9- se não retorne falso
10- fim função

fluxograma



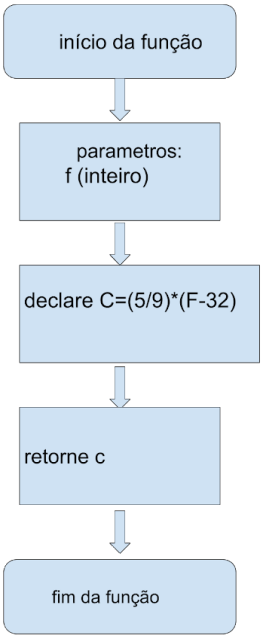
teste de mesa

passo	comando	frase	inverso	ncaracteres	i	j
3	i = 0	[o, l, a]	[]	3	0	
4	j = ncaracteres- 1	[o, l, a]	[]	3	0	2
5.1	frase[i] <- inverso[j]	[o, l, a]	[a]	3	0	2
5.2	i <- i + 1	[o, l, a]	[a]	3	1	2
5.3	j <- j - 1	[o, l, a]	[a]	3	1	1
5.1	frase[i] <- inverso[j]	[o, l, a]	[a, l]	3	1	1
5.2	i <- i + 1	[o, l, a]	[a, l]	3	2	1
5.3	j <- j - 1	[o, l, a]	[a, l]	3	2	0
5.1	frase[i] <- inverso[j]	[o, l, a]	[a, l, o]	3	2	0
5.2	i <- i + 1	[o, l, a]	[a, l, o]	3	3	0
5.3	j <- j - 1	[o, l, a]	[a, l, o]	3	3	-1

7)
1- função conversor
2- parâmetro: f tipo inteiro
3- declare C=(5/9)*(F-32)
4- retorna c
5- fim conversor

linguagem natural
1- declarar a função
2- declarar o parametro tipo inteiro
3- declare uma variável recebendo a conversão
4- retorne a variável
5- fim função

fluxograma



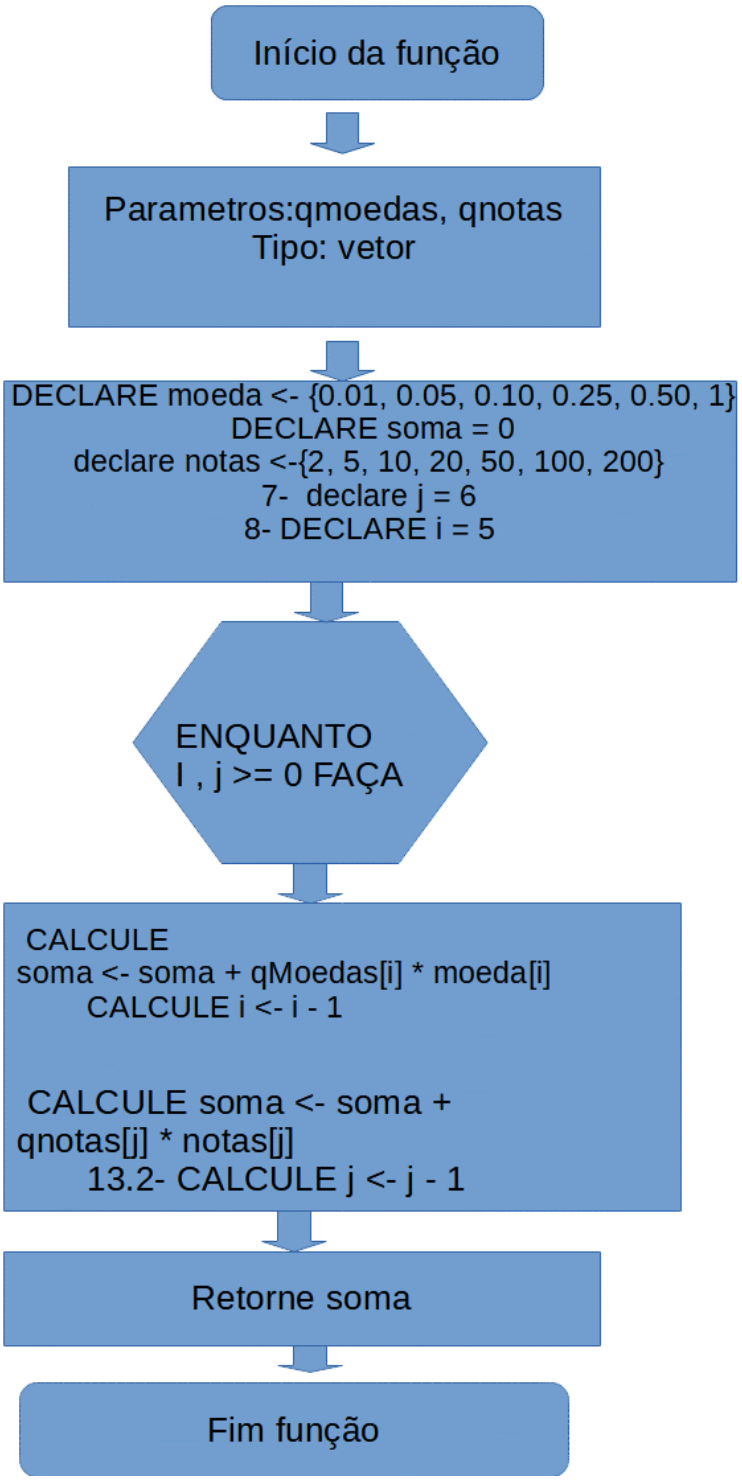
teste de mesa

passo	comando	variáveis	
		f	celsius
		212	
3	celsius (5/9*(f-32)	212	100

8)
1- Função CONTADINHEIRO
2- PARAMETROS: qMoedas, qNotas TIPO vetor
3- DECLARE moeda <- {0.01, 0.05, 0.10, 0.25, 0.50, 1}
5- DECLARE soma = 0
6- declare notas <-{2, 5, 10, 20, 50, 100, 200}
7- declare j = 6
8- DECLARE i = 5
9- ENQUANTO i >= 0 FAÇA
    8.1- CALCULE soma <- soma + qMoedas[i] * moeda[i]
    8.2- CALCULE i <- i - 1
10- FIM FAÇA
11- ENQUANTO j >= 0 FAÇA
    13.1- CALCULE soma <- soma + qnotas[j] * notas[j]
    13.2- CALCULE j <- j - 1
12- FIM FAÇA
13- retorne soma
14- fim

linguagem natural
1- declarar os parâmetros para quantidade de moedas e notas tipo vetor
2- declarar os valores da moeda
3- declarar os valores das notas
4- declarar um contador para as moedas
5- declarar um contador para as notas
6- enquanto os cantadores forem menor ou igual a 0
7- calcular soma sendo soma + quantidades de moedas * moedas
8- calcular soma sendo soma + quantidades de notas* notas
9- retornar soma
10- fim função

fluxograma


Início da função
Parametros:qmoedas, qnotas
Tipo: vetor
DECLARE moeda <- {0.01, 0.05, 0.10, 0.25, 0.50, 1}
DECLARE soma = 0
declare notas <-{2, 5, 10, 20, 50, 100, 200}
7- declare j = 6
8- DECLARE i = 5
ENQUANTO
I , j >= 0 FAÇA
CALCULE
soma <- soma + qMoedas[i] * moeda[i]
CALCULE i <- i - 1
CALCULE soma <- soma +
qnotas[j] * notas[j]
13.2- CALCULE j <- j - 1
Retorne soma
Fim função

teste de mesa

passo	comando	qmoedas	qnotas	moeda	notas	total	i	j
3	moeda <- {0.01, 0.05, 0.10, 0.25, 0.50, 1}	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}		0		6
4	notas <- {2, 5, 10, 20, 50, 100, 200}	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	0		6
5	total = 0	{ 0, 0, 0, 2, 0, 3}		{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	0		6
6	i = 5	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3	5	6
7	j = 6	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3	5	6
8.1	soma <- soma + qMoedas[i] * moeda[i]	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3	5	6
8.2	i <- i - 1	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	4	6
8.1	soma <- soma + qMoedas[i] * moeda[i]	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	4	6
8.2	i <- i - 1	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	3	6
8.1	soma <- soma + qM o e d a s [i] * m o e d a [i]	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	3	6
8.2	i < - i - 1	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	2	6
8.1	s o m a < - s o m a + q M o e d a s [i] * m o e d a [i]	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	2	6
8.2	i < - i - 1	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	1	6
8.1	s o m a < - s	{ 0, 0, 0,	{0, 0, 0,	{0.01, 0.05,	{2, 5, 10,	3,5	1	6

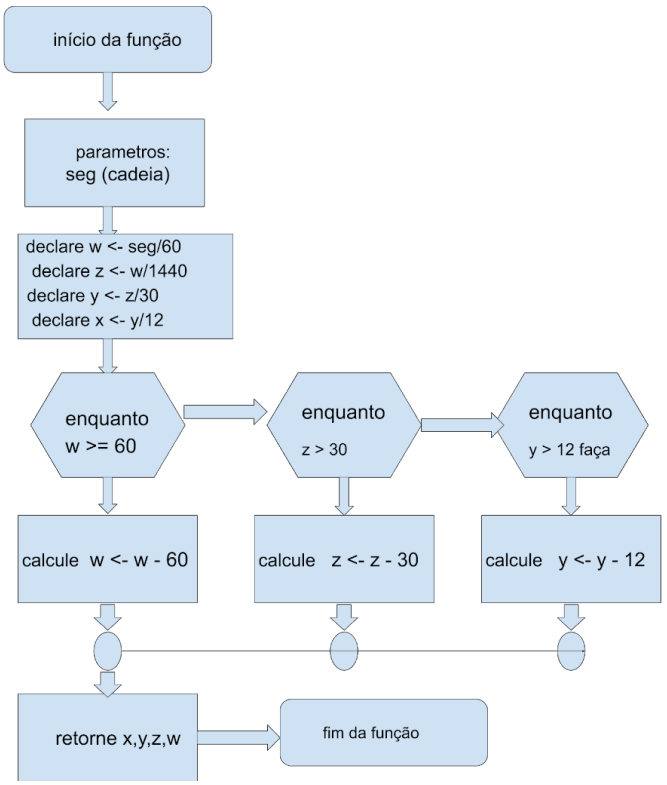
	o m a + q M o e d a s [i] * m o e d a [i]	2, 0, 3}	4, 0, 2, 1}	0.10, 0.25, 0.50, 1}	20, 50, 100, 200}			
8.2	i < - i - 1	{ 0, 0, 0, 2, 0, 3}	{0, 0, 0, 4, 0, 2, 1}	{0.01, 0.05, 0.10, 0.25, 0.50, 1}	{2, 5, 10, 20, 50, 100, 200}	3,5	0	6
10.1	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	2 0 3, 5	0	6
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	2 0 3, 5	0	5
10.1	j < - j - 1	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	4 0 3, 5	0	5
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	4 0 3, 5	0	4
10.1	j < - j - 1	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	4 0 3, 5	0	4
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	4 0 3, 5	0	3
10.1	j < - j - 1	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	3
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	2
10.1	j < - j - 1	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	2
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	1
10.1	j < - j - 1	{ 0, 0, 0, 2, 0,	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25,	{ 2, 5, 1 0, 20, 50,	463, 5	0	1

		3}		0.50, 1}	100, 200 }			
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	0
10.1	j < - j - 1	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	0
10.2	s o m a < - s o m a + q N o t a s [j] * n o t a s [j]	{ 0, 0, 0, 2, 0, 3}	{ 0, 0, 0, 4, 0, 2, 1}	{ 0.0 1, 0.0 5, 0.10, 0.25, 0.50, 1}	{ 2, 5, 1 0, 20, 50, 100, 200 }	463, 5	0	-1

9)
1- função conversor
2- parâmetro: seg tipo inteiro
3- declare w <- seg/60
4- declare z <- w/1440
5- declare y <- z/30
6- declare x <- y/12
7- enquanto w >= 60 faça
w <- w - 60
8- fim faça
9- enquanto z > 30 faça
z <- z - 30
10- fim faça
11- enquanto y > 12 faça
y <- y - 12
12- fim faça
13- declare tempo = minutos + " " + dias + "/" + meses + "/" + anos
14- retorne tempo
14- fim conversor

linguagem natural
1- declarar a função
2- declarar o parâmetro
3- declarar a varável w valendo segundos/60
4- declarar a variável z valendo minutos/1440
5- declarar a variável y valendo dias/30
6- declarar a variável x valendo meses/12
7- fazer um loop para cada variável para colocar um limite
8- declare a variável tempo igual a minutos,dias,meses e anos
9- retorne a variável tempo
10- fim da função

fluxograma



teste de mes

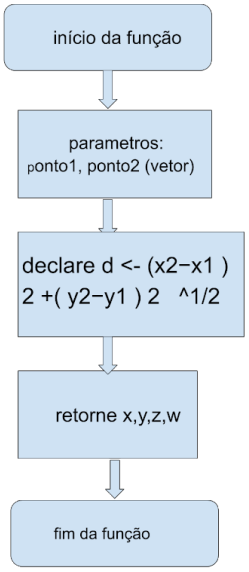
passo	comando	segundos	minutos	dias	meses	anos	tempo
3	minutos = segundos / 60	40636200	677150	470			
4	dias = minutos / 1440	40636200	677150	470			
5	meses = dias / 30	40636200	677150	470	15		
6	minutos <- minutos - 60	40636200	677150	470	15	1	
7.1	minutos <- minutos - 60	40636200	677150	470	15	1	
7.1	minutos <- minutos - 60	40636200	677150	470	15	1	
7.1	…..	40636200	…..	470	15	1	
7.1	minutos <- minutos - 60	40636200	50	470	15	1	
9.1	dias <- dias - 30	40636200	50	440	15	1	
9.1	dias <- dias -	4063620	50	410	15	1	

	30	0					
9.1	…...	40636200	50	….	15	1	
9.1	dias <- dias - 30	40636200	50	20	15	1	
10.1	meses <- meses - 12	40636200	50	20	3	1	
13	tempo = minutos + " " + dias + "/" + meses + "/" + anos	40636200	50	20	3	1	50 20/3/1

10)
1- função distância
2- parametros: ponto1, ponto2 tipo vetor
3- declare d <- (x2−x1) 2 +(y2−y1) 2 ^1/2
4- retorne d
5- fim

linguagem natural
1- declare a função
2- declarar parâmetros tipo vetor
3- declare uma variável recebendo o cálculo da distância
4- retorne essa variável
5- fim

fluxograma



teste de mesa

passo	comando	ponto1	ponto2	d
3	d = (((ponto2[0]-ponto1[0])^2) + ((ponto2[1]-ponto1[1])^2))^(1/2)	[12, 28]	[15, 32]	5