Trabalho de Testes de Software
Parte 1

Grupo:
Vitor Silva Costa - DRE: 110092602
Saulo Santos Rocha Lima - DRE: 111222933
Marcos Guimarães Antunes - DRE: 111476029
Antonio Constantino Júnior - DRE: 110177949

Particionamento

Problema: Letras

entradas: 1 character, 1 string

Interface approach
***cada particionamento tem um bloco true e outro false**

c1: primeira entrada é um character

c2: segunda entrada é uma string

c3: segunda entrada tem 1 character

c4: segunda entrada tem 1000 character

c5: segunda entrada tem entre 1 e 1000 character

c6: segunda entrada tem 0 character

c7: segunda entrada tem mais de 1000 character

c8: segunda entrada tem 0 character branco consecutivo

c9: segunda entrada tem 1 character branco consecutivo

c10: segunda entrada tem mais de 1 character branco consecutivo

c11: segunda entrada tem somente letras minúsculas

c12: segunda entrada tem somente letras minúsculas e espaços em branco

c13: primeira entrada é um character correspondente à uma letra minúscula

c14: primeira entrada é um character correspondente à uma letra maiúscula

Requisitos: (vamos utilizar o each-choice criteria)
Como temos muitos particionamentos, vamos usar each-choice. Caso contrário acabaríamos tendo um número muito grande de requisitos e, consequentemente, de testes para testar uma simples função.
TRec = {c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, not c1, not c2, not c3, not c4, not c5, not c6, not c7, not c8, not c9, not c10, not c11, not c12, not c13, not c14}

Testes:
T1 = (p, p)
requisitos cobertos: c1, c2, c3, not c4, c5, not c6, not c7, c8, not c9, not c10, c11, not c12, c13, not c14

T2 = (P, “P”+” ”+”a”+” ”+”a”*1000)
requisitos cobertos: c1, c2, not c3, not c4, not c5, not c6, c7, not c8, c9, c10, not c11, not c12, not c13, c14

T3 = (9, “”)
requisitos cobertos: not c1, c2, not c3, not c4, not c5, c6, not c7, c8, not c9, not c10, not c11, not c12, not c13, not c14

*Esse teste nao é possível ser feito em c++ devido à tipagem dos parâmetros
T4 = (a, boolean True)
requisitos cobertos: c1, not c2, not c3, not c4, not c5, not c6, not c7, not c8, not c9, not c10, not c11, not c12, c13, not c14

T5 = (a, “a”+” ”+”a”*998)
requisitos cobertos: c1, c2, not c3, c4, not c5, not c6, not c7, not c8, c9, not c10, c11, c12, c13, not c14

Requisitos/Teste	T1	T2	T3	T4	T5
c1	x	x		x	x
c2	x	x	x		x
c3	x				
c4					x
c5	x				
c6			x		
c7		x			
c8	x		x		
c9		x			x
c10		x			
c11	x				x
c12					x
c13	x				x
c14		x			
not c1			x		
not c2				x	
not c3		x	x	x	x
not c4	x	x	x	x	
not c5		x	x	x	x
not c6	x	x		x	x
not c7	x		x	x	x
not c8		x		x	x
not c9	x		x	x	
not c10	x		x	x	x
not c11	x	x	x	x	
not c12	x	x	x	x	

not c13	x	x	x	x	
not c14	x		x	x	x

Functional approach

p = porcentagem de palavras
n = número de palavras no texto

c1: p
b11: p=0; b12: 0 < p < 100; b13: p=100

c2: n
b21: n=1; b22: n=2; b23: n>2

Requisitos: (vamos usar pair-wise criteria, que nesse caso fica igual ao all combinations) Escolhemos pair-wise pois temos poucos particionamentos. Dessa forma teremos mais requisitos, resultando em mais testes que cobrirão mais possíveis casos de falha. O ganho que teremos nos testes compensará o trabalho extra causado pelo pair-wise.

TRpw = {(b11.b21), (b11.b22), (b11.b23), (b12.b21), (b12.b22) ,(b12.b23) ,(b13.b21) ,(b13.b22) ,(b13.b23)}

Testes:
T1 = (a, roxo)

T2 = (o, casa casa)

T3 = (o, casa casa casa)

T4 = (o, no mar)

T5 = (o, cai no mar azul)

T6 = (p, papagaio)

T7 = (p, papagaio preto)

T8 = (a, casa casa casa)

Requisitos/Teste	T1	T2	T3	T4	T5	T6	T7	T8
b11.b21	x							
b11.b22		x						
b11.b23			x					
b12.b21				x				
b12.b22								
b12.b23					x			
b13.b21						x		
b13.b22							x	
b13.b23								x

O requisito (b12.b21) não pode ser alcançado porque não é possível ter porcentagem entre 0 e 100 tendo apenas uma palavra no texto.

Problema: Banda

entradas: 1 número inteiro N, 1 número inteiro M, M conjuntos (X, Y, Z)

Interface approach
***cada particionamento tem um bloco true e outro false**

c1: N = 3

c2: 3 < N < 100

c3: N = 100

c4: M = 0

c5: 0 < M < 10000

c6: M = 10000

*c7: M < N * (N-1)

*c8: X <= N

*c9: Y <= N

*c10: X != Y

c11: Z = 1

c12: 1 < Z < 100

c13: Z = 100

*c14: conjunto (X, Y) único dentro dos M conjuntos (X, Y, Z)

Requisitos: (vamos usar o each-choice)
Como temos muitos particionamentos, vamos usar each-choice. Caso contrário acabaríamos tendo um número muito grande de requisitos e, consequentemente, de testes para testar uma simples função.
***os particionamentos marcados com (*) não podem ser implementados em um teste sem que se tenha o código ou sem que se saiba o comportamento esperado para tal erro.**
TRec = {c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11, c12, c13, c14, not c1, not c2, not c3, not c4, not c5, not c6, not c7, not c8, not c9, not c10, not c11, not c12, not c13, not c14}

Testes:

T1 = (3, 3, (1,2,50
2,3,100
3,1,1))

T2 = (4, 0, ())

T3 = (100, 10000, (100,99,100
2,5,70
3,1,1(*9998)))

Requisitos/Teste	T1	T2	T3
c1	x		
c2		x	
c3			x
c4		x	
c5	x		
c6			x
c7	x	x	
c8	x		x
c9	x		x
c10	x		
c11	x		x
c12	x		x
c13	x		x
c14	x		
not c1		x	x
not c2	x		x
not c3	x	x	
not c4	x		x
not c5		x	x
not c6	x	x	
not c7			x
not c8		x	
not c9		x	
not c10		x	x
not c11		x	
not c12		x	
not c13		x	
not c14		x	x

Functional Approach:

(a, b, c) = 3 músicos selecionados

c1: a
b11: a=1; b12: 1<a<N; b13: a=N

c2: b
b21: b=1; b22: 1<b<N; b23: b=N

c3: c
b31: c=1; b32: 1<c<N; b33: c=N

Requisitos: (vamos usar each-choice)
Como temos muitos particionamentos, vamos usar each-choice. Caso contrário acabaríamos tendo um número muito grande de requisitos e, consequentemente, de testes para testar uma simples função.

TRec = { b11, b12, b13, b21, b22, b23, b31, b32, b33 }

Testes:

T1 = (3, 1, (1,2,30))

T2 = (3, 1, (2,1,30))

T3 = (3, 1, (3,2,30))

T4 = (3, 1, (1,3,30))

Requisitos/Teste	T1	T2	T3	T4
b11	x			x
b12		x		
b13			x	
b21		x		
b22	x		x	
b23				x
b31			x	
b32				x
b33	x	x		