Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Projeto e Desenvolvimento de Algoritmos
AD2 2° semestre de 2016

Nome –

Assinatura –

1ª questão (valor 5.0)

Pedra-papel-tesoura-lagarto-Spock é uma extensão do clássico método de seleção do jogo pedra-papel-tesoura. A extensão segue o mesmo princípio básico do jogo original, mas inclui outras duas armas adicionais: o lagarto (formado pela mão igual a uma boca de fantoche) e Spock (formada pela saudação dos vulcanos em Star Trek). Isso reduz as chances de uma rodada terminar em um empate. O jogo foi inventado por Sam Kass e Karen Bryla, como "Rock Paper Scissors Lizard Spock".

Regras

As regras de Pedra-papel-tesoura-lagarto-Spock são:

- Tesoura corta papel
- Papel cobre pedra
- Pedra esmaga lagarto
- Lagarto envenena Spock
- Spock esmaga (ou derrete) tesoura
- Tesoura decapita lagarto
- Lagarto come papel
- Papel contesta Spock
- Spock vaporiza pedra
- Pedra quebra tesoura

Sua tarefa: Escreva um algoritmo que, dadas as entradas descritas a seguir, gere a sequencia de saídas corretas.

Entrada

A primeira linha contém um inteiro `T` (`T ≤ 100`) indicando o número de casos de teste. Cada caso de teste é descrito em uma linha separada. Cada linha contém as escolhas do jogador 1 e do jogador 2 separadas por um espaço em branco. As opções possíveis são: pedra, papel, tesoura, lagarto e Spock.

Saídas

Para cada caso de teste seu algoritmo deve gerar uma única linha com a seguinte mensagem: `"Caso #i: Jogador j"`, onde `i` é o número do caso de teste e `j` é o número do jogador vencedor, ou `"Caso #i: Empate"`, em caso de empate.

Exemplo:

No exemplo a seguir as saídas geradas pelo seu algoritmo são mostradas em vermelho e em itálico.

```
3
papel pedra
Caso #1: Jogador 1
lagarto tesoura
Caso #2: Jogador 2
Spock Spock
Caso #3: Empate
```

2ª questão (valor 5.0)

Considere uma expressão aritmética qualquer com parênteses. Um dos indicativos de erro na definição da expressão é o balanceamento dos parênteses. Para uma expressão estar balanceada quanto aos parênteses algumas regras devem ser seguidas. Por exemplo, todo parênteses que fecha deve ter um outro parênteses correspondente que abre. Não pode haver parênteses que fecha sem um prévio parênteses que abre e a quantidade total de parênteses que abre deve ser igual a quantidade que fecha.

Sua tarefa: Escreva um algoritmo que leia uma expressão com no máximo 80 caracteres e conte quantos pares abre-fecha parênteses estão validamente definidos, conforme as regras acima. Observe que pares de parênteses podem estar aninhados dentro de outros pares. Importante: só os caracteres abre e fecha parênteses devem ser analisados, o restante da expressão não deve ser considerado. Note que estamos interessados em saber quantos pares são válidos de acordo com as regras e não se a expressão toda está correta.

Exemplo:

`(a+(b*c)-(2-a))`	3 pares: `(b * c)`, `(2 - a)` e o par mais externo `(a+(b*c)-(2-a))`
`(a+b*(2-c)-2+a)*2`	2 pares: `(2 - c)` e o mais externo `(a+b*(2-c)-2+a)`
`(a*(b)-(2+c)`	2 pares: `(b)` e `(2 + c)`
`2*(3-a))`	1 par: `(3 - a)`
`)3+b*(2-c)(`	1 par: `(2 - c)`

Entrada

Como entrada, haverá uma expressão com até 80 caracteres.

Saída

A saída deve indicar quantos pares abre fecha parênteses válidos foram encontrados na expressão. Exemplo da saída gerada pelo seu programa (em vermelho e itálico):

```
(a+(b*c)-(2-a))
3
```

Para resolver este problema considere que estão disponíveis as seguintes funções e procedimentos:

procedimento le(**saídas:** vetor)

Lê um conjunto de caracteres do teclado e o armazena em `vetor`.

função tamanho(**entradas:** str)

Retorna o número de caracteres na cadeia de caracteres passada como parâmetro

Exemplo:

```
imprima tamanho('CEDERJ')   # imprimiria 6
```

função charAt(**entradas:** str, pos)

Retorna o caractere na posição **pos** da cadeia de caracteres **str** passada como parâmetro.

Exemplo:

```
imprima charAt('CEDERJ', 3)      # imprimiria 'D'
```