Lista de exercícios #7 – manipulando strings

Sobre a lista:

Aqui a nossa linguagem algorítmica estruturada de alto nível ganha novos tipos de dados: *strings*, para melhor representar algumas situações-problema da vida real.

1. [*palavra palíndrome*] Uma palavra é dita *palíndrome* se ele puder ser lida da mesma forma nos dois sentidos de escrita. Faça um algoritmo que leia uma palavra do usuário e diga se ela é ou não é palíndrome.

2. [*razão vogais consoantes*] Faça um algoritmo que obtenha a razão entre o número de vogais e o número de consoantes de uma frase lida do usuário. Todos os outros caracteres que não sejam letras ou dígitos devem ser ignorados no cálculo.

3. [*quantidade de menores e maiores*] Faça um algoritmo que leia um caracter e uma string e diga quantos caracteres, dentro da string, são menores e quantos são maiores que o caracter digitado.

4. [*remove caracter da string*] Dada uma string de origem, a ser lida do usuário, construa em uma segunda variável, uma string destino que seja "igual" à origem, mas que não contenha qualquer ocorrência de um determinado caracter x, também lido do usuário. Ao final, escreva o conteúdo da variável string gerada. Considere, para simplificar, que os caracteres acentuados são diferentes dos seus correspondentes sem acentuação e que maiúsculas são diferentes das minúsculas.

5. [*verifica caracteres alternados*] Faça um algoritmo para ler uma string e verificar se cada caracter nela existente é, alternadamente, maior e menor que o anterior, escrevendo `''sucesso''` em caso afirmativo, e `''falha''` caso contrário. A string tanto pode começar por uma sequência maior-menor quanto por uma menor-maior.

6. [*retira espaços em branco*] Faça um algoritmo para ler uma frase, montar uma segunda frase que não tenha nenhum dos espaços em branco da primeira e escrevê-la na saída.

7. [*transforma espaços duplicados em simples*] Faça um algoritmo que leia uma frase, transforme todos os espaços em branco múltiplos em espaços simples (em uma nova variável) e reescreva a frase (o conteúdo dessa nova variável) na saída. A frase resultante não deve ter nenhum espaço em branco nem no seu início nem no seu final.

8. [*duplica espaços*] Faça um algoritmo para ler uma frase e montar uma segunda fazendo com que cada espaço em branco da primeira seja duplicado, escrevendo-a na saída.

9. [*abrevia nomes*] Faça um algoritmo para ler uma string, representando um nome próprio, e gere uma segunda string que represente o mesmo nome abreviado, ou seja, a string deve ser formada apenas pelos primeiros caracteres de cada nome seguido de um ponto. Lembre-se de manter apenas um espaço em branco entre as abreviaturas.

10. [*obtém subcadeia*] Faça um algoritmo que leia uma string s, um número inteiro a e um número inteiro b, e gere uma nova cadeia r, com o trecho de s, com b caracteres, que aparecem a partir da posição a. Note que não há nenhum problema se b for maior do que a quantidade de caracteres disponíveis. Escreva r na saída.

11. [*apaga subcadeia*] Faça um algoritmo que leia uma cadeia de caracteres s, um número inteiro a e um número inteiro b e remova de s todos os b caracteres que aparecem a partir da posição a da string s, guardando o resultado em uma nova string r. Não há nenhum problema se b for maior do que a quantidade de caracteres disponíveis. Escreva r na saída.

12. [*posição da subcadeia*] Faça um algoritmo que leia duas strings `frase` e `sub` e encontre a posição inicial onde `sub` aparecem em `frase`, escrevendo esse valor ao final. Se tal fato não acontece, escreva 0 na saída.

13. [*quais caracteres aparecem*] Dada uma string origem, cria uma outra string destino que seja igual à primeira, mas que não contenha duplicações dos caracteres que aparecem em origem, ou ainda, que contenha apenas os caracteres que aparecem em origem mas que não sejam repetidos. Escreva o resultado ao final.

14. [*formata um nome próprio*] Construir um algoritmo que leia uma string, supostamente um nome próprio, e converta a primeira letra de cada palavra para maiúscula e todas as outras minúsculas. Observe que a string pode estar mal scrita, com espaços em branco em quaisquer lugares.

15. [*transforma número em string hexadecimal*] Faça um algoritmo para ler um número inteiro e gerar uma string que represente o mesmo número, porém, na base hexadecimal. Escreva a string resultante na saída.

16. [*criar sequência crescente de caracteres*] Faça um algoritmo para ler dois caracteres e montar uma string que contenha, em ordem crescente, todos os caracteres a partir do primeiro até o segundo, inclusive, se e apenas se o primeiro caracter é menor ou igual ao segundo.

17. [*cifra de César*] Faça um algoritmo que codifique uma mensagem, lida do usuário, segundo a *Cifra de César*. A codificação é feita substituindo-se cada letra pela sua terceira sucessora, quanto que a decodificação é feita alterando-a pela terceira predecessora. Considere que as sucessoras das letras `'X'`, `'Y'` e `'Z'` são, respectivamente, `'A'`, `'B'` e `'C'`,e que o inverso também é verdadeiro. O algoritmo deve tratar letras maiúsculas e minúsculas iguais, ou seja, `'M'` e `'m'` são iguais.

18. [*encontra frase do código hash*] Faça um algoritmo que leia um número inteiro x e, em seguida e continuamente, leia frases do usuário e escreva, para cada uma destas frases, um valor de *hash* na saída. O valor de *hash* em questão é obtido pelo resto da divisão, por 63, do somatório dos códigos ASCII de todos os caracteres da frase. O algoritmo deve parar ao ser digitada uma frase cujo valor de *hash* seja igual ao número x digitado pelo usuário, que deve estar, obrigatoriamente, no intervalo fechado de 0 a 62.

19. [*converte string em número*] Faça um algoritmo que leia uma string que deveria conter apenas dígitos e converta-o em sua representação numérica inteira equivalente em uma variável inteira, escrevendo-a ao final. Observe que a string pode ter, sem problema, um número negativo. Qualquer caracter diferente de um dígito, dentro da cadeia, deve invalidar a operação, e uma mensagem elucidativa deve ser fornecida ao usuário ao término do programa.

20. [*distância entre caracteres*] Seja o alfabeto da língua portuguesa. Diremos que um caracter a está n posições distante de um caracter b, se existirem, no máximo $n-1$ caracteres entre os dois no alfabeto. Assim, faça um algoritmo que leia um caracter c e uma cadeia de caracteres s e diga quantos caracteres (contabilizando as eventuais duplicações), dentro de x, estão a uma distância de 4 posições de c.