Universidade Tecnológica Federal do Paraná
Departamento Acadêmico de Informática
CSF13 – Fundamentos de Programação 1
Prof. Bogdan Tomoyuki Nassu
Profa. Leyza Baldo Dorini

(***) 1. Nas últimas décadas, foram realizados inúmeros estudos envolvendo a análise da estrutura do DNA de diversas espécies. Os desafios vão do sequenciamento em si à detecção de genes associados à propensão para certas doenças; da identificação de diferenças entre espécies à determinação da ancestralidade de uma pessoa. Tais estudos só foram possíveis graças ao avanço da computação, pois envolvem o processamento de grandes massas de dados. Esses dados normalmente são representados como cadeias contendo as letras A, G, C e T, que se referem respectivamente às bases nucleotídicas adenina, guanina, citosina e timina. Por exemplo:

ACCCGGGGTTTATGA

Uma forma compacta de representar uma sequência de DNA é substituir sequências com 2 ou mais bases repetidas por um número seguido da base. A sequência do exemplo acima ficaria assim:

A3C4G3TATGA

Suponha um sistema que representa uma sequência de DNA como um vetor do tipo `byte`, onde `byte` é um valor inteiro de 1 byte. As 4 bases são representadas por valores negativos: A=-1, G=-2, C=-3, T=-4. Desta forma, a sequência do exemplo seria representada assim:

-1	-3	-3	-3	-2	-2	-2	-2	-4	-4	-4	-1	-4	-2	-1

A sequência compactada usa a mesma representação para as bases, mas com valores positivos para os contadores das bases repetidas. Ou seja, a representação do exemplo seria:

-1	3	-3	4	-2	3	-4	-1	-4	-2	-1

Segundo a especificação do sistema, o tipo `byte` é suficiente para armazenar o maior número de bases repetidas que pode ocorrer na prática, ou seja, não é preciso se preocupar com overflow.

A função a seguir deveria receber como parâmetro uma sequência de DNA (`in`) de tamanho `n`, e colocar no vetor de saída `out` uma versão comprimida da sequência. O vetor de saída tem o mesmo número de posições do vetor de entrada, mas pode usar apenas uma parte dessas posições. Por isso, a função também deveria retornar o número de posições usadas de fato no vetor de saída. Infelizmente, a função não está funcionando. O vetor de saída contém valores incorretos, e o valor de retorno é sempre 0.

```
0     int comprimeSequencia (byte* in, int n, byte* out)
1     {
2         int inicio = 0, usadas = 0, val, cont;
3
4         while (inicio < n)
5         {
6             val = in [inicio];
7             while (inicio+cont < n && in [inicio+cont] == val)
8                 cont++;
9
10            if (cont == 1)
11            {
12                out [usadas] = val;
13            }
14            else
15            {
16                out [usadas] = cont;
17                out [usadas+1] = val;
18            }
19
20            inicio += cont;
21        }
22
23        return (usadas);
24    }
```

Descreva o que precisa ser feito para que a função funcione corretamente. Não explique os erros nem reescreva o código, apenas descreva as correções de forma clara e não ambígua (ex: linha 1: mudar para..., entre as linhas 1 e 2, acrescentar...). Você deve modificar/remover/acrescentar apenas 2 ou 3 declarações. Dica: os erros são de lógica, e não envolvem a sintaxe da linguagem C nem a formatação dos dados de E/S.

(***) 2. Uma fábrica de dados (de 6 faces, para jogos!) encomendou um sistema para controle de qualidade no qual os dados que saem de uma esteira caem em uma caixa com diversos compartimentos. A caixa é posicionada de forma que cada compartimento contenha exatamente um dado. Esta caixa é girada rapidamente de forma a "rolar" mecanicamente os dados, e após cada giro, uma fotografia é tirada por uma câmera posicionada sobre a caixa. Um sistema de visão computacional identifica então o valor de cada dado. O processo é repetido 100 vezes em menos de 1 minuto. Por fim, é chamada uma função com o seguinte protótipo:

```
int analisa (int*** in, int largura, int altura, int** out);
```

O parâmetro `in` é uma matriz de entrada, já alocada dinamicamente, com o tamanho dado pelos parâmetros `largura` e `altura`. Cada posição da matriz `in` está associada a um compartimento da caixa (e, portanto, a um dado), e deve ser interpretada como um vetor de tamanho 100, com cada posição do vetor indicando o resultado obtido em uma rolagem de um dado. Por exemplo, `in[0][1][2]` indica o valor da terceira rolagem do dado que está no segundo compartimento da primeira fileira da caixa.

Um dado é testado analisando-se os resultados das suas rolagens. Observa-se quantas vezes cada um dos 6 valores foi sorteado. Se o número de ocorrências em 100 rolagens estiver no intervalo [15,18] para todos os valores, o dado é aprovado, do contrário ele é reprovado. O parâmetro de saída `out` é uma matriz com o mesmo tamanho que `in`, também já alocada. Cada posição deve ser preenchida com 1 se o dado na posição correspondente tiver sido aprovado, ou 0 do contrário.

A função retorna 1 se for detectado um erro de produção ou alinhamento da máquina, ou 0 do contrário. Um erro é detectado se pelo menos metade dos dados em uma linha ou coluna tiver sido reprovada. Valores não-inteiros são arredondados para baixo – por exemplo, se a caixa tiver 7x10 compartimentos, um erro é identificado se existir uma linha com 3 ou mais dados reprovados, ou uma coluna com 5 ou mais dados reprovados.

a) Escreva uma função que avalia as rolagens de um único dado, retornando 0 se o dado for reprovado ou 1 do contrário. A função deve receber como parâmetros apenas o vetor de valores do dado avaliado (não uma matriz!) e o seu tamanho. O protótipo da função deve ser:

```
int avaliaDado (int* rolagens, int n);
```

b) Escreva a função `analisa`. Esta função deve realizar o teste para todos os dados, e verificar também a existência de erros de alinhamento ou produção. A função deve chamar a função `avaliaDado` criada na questão a).