Aula prática - Semana 11 - ERE

Nesta lista, a questão 1 constitui a atividade avaliativa que será utilizada para contabilizar notas de atividades práticas. As demais questões são desafios que não contabilizam nota, mas cuja realização é desejável para exercitar lógica. A questão 1 possui exemplo de execução associado. Note que é apenas um exemplo, uma vez que vocês podem propor suas próprias formas de entrada e saída de dados. No entanto, é desejado que se assemelhe ao padrão apresentado no exemplo.
Os seguintes pontos serão considerados na avaliação:

- Atender os requisitos do enunciado.
- Indentação e organização de código.
- Utilização dos tipos corretos de variáveis.
- Utilização correta de constantes nomeadas quando necessário.
- Observação das boas práticas de programação vistas em aula.
- Decomposição do programa em funções
- Utilização adequada de structs
- Realização de validação quando a questão requisitar.

ATENÇÃO: O exercício abaixo tem como objetivo exercitar a ideia de representação de informações e de movimentação relacionadas ao trabalho final. Note que no exercício abaixo usamos um **sistema de coordenadas diferente** do convencional. Ou seja, neste sistema de coordenadas, o ponto mais próximo da **origem** fica no **canto superior esquerdo** e o mais distante fica no **canto inferior direito do ambiente**. Assim, as posições aumentam da **esquerda pra direita**, e de **cima pra baixo**. Ou seja, é diferente dos casos em que trabalhamos anteriormente, em que usávamos um sistema de coordenadas cartesianas convencional. Adotei essa perspectiva porque diversas bibliotecas que vocês podem usar no trabalho (como a conio) representam a tela dessa forma. Isso facilita que vocês transfiram o raciocínio deste exercício para o trabalho. É análogo ao caso de uma matriz, onde cada célula é uma posição, e o primeiro elemento está no canto superior esquerdo e o último elemento está no canto inferior direito. No entanto, é importante deixar claro que não estou pedindo para que representem visualmente essas informações na tela usando bibliotecas adicionais. Aqui só estou trabalhando conceitos de manipulação de informação de posições e direções, sem pedir que vocês representem isso visualmente.

1. Desenvolva as seguintes estruturas que representam as informações relacionadas ao trabalho final. Neste caso, consideraremos apenas informações relacionadas à posição e direção de movimentação do ninja. Considere que os ogros só podem estar posicionados em pixels específicos (posições inteiras) e só podem se mover de pixel em pixel (As **posições** e os **deslocamentos** são **inteiros**).

- Defina as estruturas abaixo (usando typedef):

- A estrutura POSICAO, representa uma posição em um plano 2D, é definida em função de dois números inteiros que representam as coordenadas x e y em que o ninja se encontra.
- A estrutura DIRECAO, representa uma direção em um plano 2D, é definida em função de dois números inteiros que representam o deslocamento no eixo x e no eixo y. Considere que o deslocamento em cada eixo pode assumir os valores -1, 0 ou 1.
- A estrutura NINJA, que representa um ninja, possui uma posição e uma direção de movimento. Use as estruturas anteriores para definir a estrutura NINJA.

- Faça uma função tipada "moveNinja" que recebe um NINJA por referência, e duas estruturas posição, que representam o canto superior esquerdo e o canto inferior direito do retângulo que define o ambiente em que o ninja vai se mover. A função deve tentar mover o ninja da posição em que ele está, para a nova posição, considerando a sua direção de movimento. Se o movimento for possível (considerando os limites do ambiente), a função **deve atualizar a posição** do ninja e **retornar 1**. Caso o movimento não seja possível (ou seja, o ninja encontrou uma borda do ambiente), a função **não deve atualizar a posição** (porque a posição não seria válida) e deve **retornar 0**. Considere que o movimento só é possível se o próximo passo leva o ninja para uma posição dentro das bordas do ambiente. Se a nova posição estiver fora dos limites do retângulo, o movimento não é possível.
- Faça um programa principal que utiliza as estruturas e a função definidas anteriormente. O programa deve ler as coordenadas dos pontos que definem o canto superior esquerdo do ambiente do ninja e o canto inferior direito do mapa. As coordenadas x e y do ponto inferior direito devem ser maiores que as coordenadas do ponto superior esquerdo. Faça consistência. O programa deve definir um ninja, e ler sua posição inicial e sua direção de movimento. A posição inicial deve estar dentro dos limites do retângulo e a direção só pode receber os valores {-1,0,1} nas coordenadas x e y. Faça consistência. Você deve garantir também que o ninja não tenha o valor 0 como deslocamento em ambos os eixos x e y (embora seja admissível o zero em um dos eixos). Considere que o valor da posição aumenta da esquerda para direita e de cima para baixo. O programa deve continuamente realizar a movimentação do ninja, imprimindo na tela a sua posição atual a cada passo. A movimentação deve ser finalizada quando o ninja não puder mais se mover naquela direção. No fim, o programa deve informar quantos passos o ninja deu até encontrar um limite.

Exemplo de execução:

```
Informe a posição do canto superior esquerdo do mapa: 0 0
Informe a posição do canto inferior direito do mapa: 10 10
Informe as coordenadas x e y da posição do ninja: 5 4
Informe os deslocamentos do ninja em x e y: 1 0
Ninja se movendo...
```

```
Posição do ninja depois do passo 1: (6,4)
Posição do ninja depois do passo 2: (7,4)
Posição do ninja depois do passo 3: (8,4)
Posição do ninja depois do passo 4: (9,4)
Posição do ninja depois do passo 4: (10,4)
O ninja deu 5 passos antes de encontrar o limite.
```

2. (Desafio) Faça uma nova versão do programa anterior. O programa deve considerar um número N_NINJAS de ninjas (use #define N_NINJAS 4) e inicializar aleatoriamente as posições e direções dos ninjas, respeitando os limites do ambiente e os valores admissíveis para direções (garanta que nenhum ninja tem zero como valor de deslocamento de ambos os eixos). O programa deve repetidamente chamar a função "moveNinja" para todos os ninjas, até que todos encontrem encontrem um limite do mapa. A cada iteração, o programa deve exibir as informações de cada ninja e indicar quando o ninja encontrou um limite.

3. (Desafio) Faça uma versão do programa anterior em que, em vez de parar quando todos os ninjas encontram obstáculos, deve ler um valor n de iterações, e deve realizar n iterações de movimentação dos ninjas. No entanto, nesta versão, quando um ninja encontra um obstáculo, o programa deve determinar uma nova direção válida (não pode ser zero em ambos os eixos) aleatória para o ninja seguir na próxima iteração. A cada iteração, o programa deve exibir as informações de cada ninja e indicar quando o ninja trocou de direção.