Lista 7 - CI055 - Algoritmos e Estruturas de Dados I

A lista abaixo deve ser resolvida de maneira individual. As soluções dos exercícios deverão ser discutidas e entregues seguindo as instruções da página da disciplina:

- Alunos da turma do **Prof. Marcos Castilho** devem consultar:
  `http://www.inf.ufpr.br/alexander/ci055/instrucoes-turma-marcos.html`
- Alunos da turma do **Prof. Daniel Weingaertner** devem consultar:
  `http://www.inf.ufpr.br/alexander/ci055/instrucoes-turma-daniel.html`

O prazo para entrega desta lista é: **24 de Junho de 2015 (quarta-feira).**

Enunciados dos exercícios

Esta lista possui um único exercício, o qual foi projetado para ser resolvido durante o período de uma semana. Ele adiciona caraterísticas aos exercícios já resolvidos em sala de aula que adotam reduções do tamanho da memória da estrutura de dados quando o objeto principal é uma matriz espersa. Veja o exemplo de código que decide se nenhuma das n Torres de um tabuleiro de xadrez de n por n ataca a outra. Nesse programa fonte, o tabuleiro que tem $n \times n$ casas é uma matriz esparsa, a qual foi representada na forma de um vetor de inteiros onde a dezenda de cada inteiro guarda a linha onde há uma Torre e a unidade guarda a coluna.

`http://www.inf.ufpr.br/cursos/ci055/ntorres.pas`

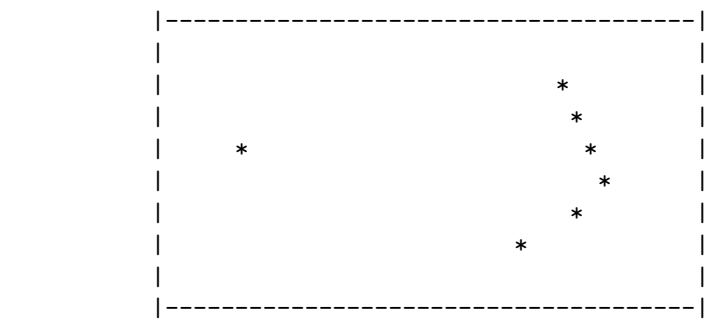
Agora, neste exercício, além de lidar com o mesmo conceito de matrizes esparsas, os primeiros passos de simulação computacional serão dados por meio da animação aproximada do movimento da bola branca em uma mesa de sinuca. Paratanto, vamos precisar de alguns conceitos da Geometria, da Trigonometria e, claro, da Física. Todavia, atenção: nesta simulação, a noção de tempo não é fundamentamente o da Física mas, sim, o tempo discreto de um ciclo de re-cálculo da posição da bola branca. Isso tem implicações sobre o retrato da mesa em cada um dos quadros da simulação, os quais não são função do tempo Físico.

Adicionalmente, neste mundo ideal, não há colisão entre bolas na primeira versão do programa. Além disso, apenas a bola branca se movimenta e colide contra as laterais da mesa, seguindo então uma trajetória de um ângulo reflexivo de acordo com as leis de Física.

Fazer um programa em Pascal para ler, da entrada padrão (*i.e.*, teclado), os dados de dimensões de uma mesa de sinuca, a quantidade de bolas (incluindo a bola branca) e a localização linha e coluna de cada bola (iniciando com a da bola branca).

O formato da impressão do programa, sempre na saída padrão (*i.e.*, monitor de vídeo), deverá ser semelhante ao de uma mesa de sinuca disposta com a Largura na orientação vertical e o Comprimento na horizontal. Incluindo as bordas, tanto para a leitura como para a impressão, a Largura (em Linhas) e o Comprimento (em Colunas) são limitadas a 40 unidades da tela (chamadas aqui de pixels). A máxima quantidade de bolas é de 9.

Além desses dados, mais dois são lidos ainda: a norma do vetor velocidade inicial (em pixels por segundo) e o ângulo de movimento da bola branca (em graus). As medidas de ângulo são de valores reais maiores ou iguais a 0 (zero) ou menores do que 360. A orientação positiva do ângulo é no sentido antihorário (da Trigonometria) e com seu vértice de medição centrado na bola.



Veja uma forma aproximada da saída padrão para o exemplo acima, em que a mesa de sinuca tem Largura = 10 pixels, Comprimento = 40 pixels e a quantidade de 7 bolas.

Após a leitura, o programa deve dar início ao procedimento de simulação. Tente executar uma solução já implementada para ver um comportamento aproximado do seu programa. Primeiro, baixe os seguintes arquivos para o seu diretório de trabalho:

`http://www.inf.ufpr.br/cursos/ci055/mesasinuca`
`http://www.inf.ufpr.br/cursos/ci055/dadossinuca.txt`

O arquivo **mesasinuca** não é um programa fonte mas apenas um código executável em Linux. Por isso, mude a permissão do arquivo **mesasinuca** para ele ficar executável e, em seguida, basta executar assim:

```
chmod u+rx mesasinuca
./mesasinuca < dadossinuca.txt
```

O arquivo **dadossinuca.txt** tem todos os dados de entrada. Para o exemplo anterior, veja que ele contém:

```
cat dadossinuca.txt
10 40 7 5 7 3 30 4 31 5 32 6 33 7 31 8 27 2 325
```

Se o programa **mesasinuca** for executado sem a entrada do arquivo, o aspecto da saída/entrada padrão será:

```
./mesasinuca
Entre com a Largura da mesa em pixels (no maximo 40): 10
Entre com o Comprimento da mesa em pixels (no maximo 40): 40
Entre com a quantidade de Bolas da mesa (no maximo 9): 7
Entre com a posicao da Largura da Bola Branca: 5
Entre com a posicao do Comprimento da Bola Branca: 7
Entre com a posicao da Largura da Bola 2: 3
Entre com a posicao do Comprimento da Bola 2: 30
Entre com a posicao da Largura da Bola 3: 4
Entre com a posicao do Comprimento da Bola 3: 31
Entre com a posicao da Largura da Bola 4: 5
Entre com a posicao do Comprimento da Bola 4: 32
Entre com a posicao da Largura da Bola 5: 6
Entre com a posicao do Comprimento da Bola 5: 33
Entre com a posicao da Largura da Bola 6: 7
Entre com a posicao do Comprimento da Bola 6: 31
Entre com a posicao da Largura da Bola 7: 8
Entre com a posicao do Comprimento da Bola 7: 27
Entre com a Velocidade Inicial da Bola Branca em pixels/s (de 0.01 a 3.0): 2
Entre com o Angulo Inicial da Bola Branca em graus (de 0 a 359): 325
```

Crie outros exemplos de entrada para inspecionar o comportamento aproximado do programa. Nesta primeira versão, não se preocupe tanto com as leis da Física para dar realismo à queda de velocidade da bola branca ao longo do tempo. Qualquer tipo de frenagem será boa. Para facilitar a sua solução, use os seguintes recursos:

- Os procedimentos de leitura e impressão aproximados do problema das n-Torres;
- As funções trigonométricas **sin** (seno) e **cos** (coseno), mas cuidado pois os argumentos precisam ser passados em radianos quando a entrada de dados deste programa deve ser fornecida em graus;
- Use a biblioteca do **fpc** chamada de **crt** pois com ela você poderá usar as funções **clrscr** (limpa a tela), **delay** (que retarda o processamento em um certo tempo passado em milisegundos).

Uma possível composição do corpo de comandos do Programa Principal poderia ser o seguinte:

```
program mesa_de_sinuca;
uses crt;
const
  maxbolas = 9;
  PI = 3.14159265;
type
  vetor = array[1..maxbolas] of real;
var
  L, C, B: integer;
  Vel, Ang: real;
  mesa: vetor;

     ...
     ...

begin
```

```
  write('Entre com a Largura da mesa em pixels (no maximo 40): ');
  read(L);
  write('Entre com o Comprimento da mesa em pixels (no maximo 40): ');
  read(C);
  write('Entre com a quantidade de Bolas da mesa (no maximo 9): ');
  read(B);
  le_posicao_das_bolas(mesa, L, C, B);
  write('Entre com a Velocidade Inicial da Bola Branca em pixels/s (de 0.01 a 3.0): ');
  read(Vel);
  write('Entre com o Angulo Inicial da Bola Branca em graus (de 0 a 359): ');
  read(Ang);
  clrscr;
  imprime_mesa(mesa, L, C, B);
  simula(mesa, L, C, B, Vel, Ang);
end.
```

Finalmente, se houver tempo, como sugestão, tente inventar uma forma de colisão entre bolas para que mais de uma delas esteja em movimento em um dado instante. Talvez seja bom usar a função **random** para gerar algum tipo de impressão de dispesão das bolas nas colisões. Tente também alterar a representação da matriz esparsa para receber mais do que 9 bolas na mesa.