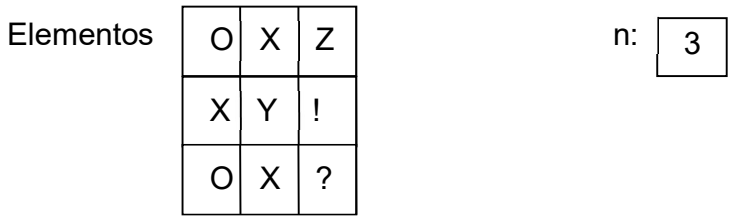
Universidade Federal de Minas Gerais **Algoritmos e Estruturas de Dados II**
Instituto de Ciências Exatas **2º Semestre de 2018**
Departamento de Ciência da Computação

Trabalho Prático 1 – Revisão de Programação e Tipos Abstratos de Dados

Valor: 10 pontos
Data de devolução: 13/09/2018

O objetivo desse trabalho é ambientar o aluno com a linguagem de programação C e explorar os conceitos de Tipos Abstratos de Dados (TADs) e análise de complexidade.

Você deverá implementar um tipo abstrato de dados **`Tabuleiro`** para representar uma matriz quadrada onde cada elemento é do tipo caractere. Seu tipo abstrato deverá armazenar os caracteres do tabuleiro (elementos da matriz) e a dimensão da matriz *n*. Considere que o tamanho máximo permitido para o tabuleiro é 10 (não é necessário utilizar alocação dinâmica de memória). A figura abaixo mostra um exemplo da estrutura:



As operações que devem ser implementadas em seu TAD são

1. Definir o tamanho do tabuleiro que vai ser utilizado
   `void SetTamanho(Tabuleiro *T, int n);`
2. Colocar um elemento c em uma posição (i,j) do tabuleiro.
   `void SetElemento(Tabuleiro *T, int i, int j, char c);`
3. Recuperar o elemento de uma posição (i,j) o tabuleiro:
   `char GetElemento(Tabuleiro T, int i, int j);`
4. Colocar um mesmo elemento c em todas as posições do tabuleiro.
   `void MarcaTodos(Tabuleiro *T, char c);`
5. Testar se uma linha i do tabuleiro possui todos os elementos iguais a c, retornando *true* (1) ou *false* (0)
   `int TestaLinha(Tabuleiro T, int i, char c);`
6. Testar se uma coluna j do tabuleiro possui todos os elementos iguais a c, retornando *true* (1) ou *false* (0)
   `int TestaColuna(Tabuleiro T, int j, char c);`
7. Testar se a diagonal esq-dir possui todos os elementos iguais a c, retornando *true* (1) ou *false* (0)
   `int TestaDiagonalED(Tabuleiro T, char c);`
8. Testar se a diagonal dir-esq possui todos os elementos iguais a c, retornando *true* (1) ou *false* (0)
   `int TestaDiagonalDE(Tabuleiro T, char c);`
9. Testar se uma linha i do tabuleiro possui todos os elementos diferentes, retornando *true* (1) ou *false* (0)
   `int TestaDiferente(Tabuleiro T, int i);`
10. Imprimir tabuleiro
    `void Imprime(Tabuleiro T);`

Implemente o seu TAD em arquivos separados do programa principal (`Tabuleiro.c e Tabuleiro.h`). Se necessário, você pode criar outras funções auxiliares em seu TAD. Suas funções devem executar testes de consistência (ex:, não se pode colocar um elemento na posição (6,6) se o tabuleiro foi configurado com n= 5).

Uma vez criado o seu TAD, você deverá utilizá-lo em **dois programas diferentes**.

O primeiro programa é um *Caça Níqueis* e deve se chamar `CN`. Nesse programa, o tabuleiro deve ter tamanho ímpar e ser preenchido com os elementos '#', '@', '%', '?' e '*'. Caso uma das duas diagonais tenha todos os elementos iguais, ou caso todos os elementos da linha central sejam diferentes uns dos outros, seu programa deve imprimir "**Bingo!!!**". Senão deve imprimir "**Tente outra vez...**". O tabuleiro deve ser impresso nos dois casos antes da mensagem. Você deve testar duas formas de preenchimento do tabuleiro: a primeira sorteando os elementos aleatoriamente e a segunda lendo os elementos de um arquivo, A passagem desses parâmetros deve ser feita através da linha de comando (parâmetros `argc` e `argv` vistos em sala). Se apenas o tamanho do tabuleiro for passado como parâmetro, o preenchimento deve ser aleatório. Se forem passados o tamanho e o nome do arquivo, os dados devem ser lidos do arquivo. Por exemplo:

```
% CN 5              (executa o programa com um tabuleiro de tamanho 5 preenchido aleatoriamente)
% CN 3 dados.txt (executa o programa com o tabuleiro de tamanho 3 e dados lidos do arquivo dados.txt)
```

O arquivo contém os elementos de cada linha do tabuleiro em cada linha do arquivo separado por espaços, por exemplo:

```
@ @ %
? % @
# # ?
```

O segundo programa é o tradicional *Jogo da Velha* e deve se chamar `Velha`. Os jogadores 'X' e 'O' vão jogando alternadamente até que um deles ganhe ou o jogo termine em "velha". No primeiro caso, deve ser impresso: "**O vencedor foi X!**" ou "**O vencedor foi O!**" e no segundo caso "**Deu Velha!**". A cada jogada, o seu programa deverá solicitar a linha e coluna que o jogador deseja marcar e imprimir o tabuleiro atualizado (considere linhas e colunas variando de 0 a 2, e use o símbolo _ para as casas não preenchidas). Testes de validação (jogadas repetidas, ou fora do tabuleiro devem ser testadas). O seu programa deverá ter três opções: Na primeira, o computador joga de forma aleatória contra você; Na segunda opção, dois jogadores humanos jogam um contra o outro; Por fim, na terceira opção, as jogadas devem ser lidas de um arquivo (nesse caso, você não precisa imprimir o tabuleiro a cada jogada, apenas o tabuleiro e resultado final). Aqui também será utilizada a leitura de parâmetros via linha de comando:

```
% Velha 1              (modo com um jogador x computador)
% Velha 2              (modo com dois jogadores alternados)
% Velha 3 dados.txt    (modo com jogadas lidas do arquivo)
```

O Arquivo deverá conter uma jogada (linha e coluna) por linha do arquivo separadas por espaço. Por exemplo:

```
0 0
0 1
1 1
0 2
2 2
```

Nesse caso, deverá ser impresso:

```
X O O
_ X _
_ _ X
O vencedor foi X!
```

(**Note que os seus programas principais não devem acessar diretamente a estrutura interna do TAD. Se necessário, acrescente novas funções ao seu TAD detalhando-as na documentação do trabalho**)

O que deve ser entregue:

- Códigos fonte dos programas e do TAD (bem identados e comentados).
- Documentação do trabalho. Entre outras coisas, a documentação deve conter:
  1. Introdução: descrição do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. Implementação: descrição sobre a implementação do programa. Deve ser detalhada a estrutura de dados utilizada (de preferência com diagramas ilustrativos), o funcionamento das principais funções e procedimentos utilizados, o formato de entrada e saída de dados, compilador utilizado, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado.
  3. Estudo de Complexidade: estudo da complexidade do tempo de execução dos procedimentos implementados e do programa como um todo (notação O), considerando tabuleiros de tamanho *n*.
  4. Conclusão: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  5. Bibliografia: bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso

Obs1: Apesar desse trabalho ser relativamente simples, a documentação pedida segue o formato da documentação que deverá ser entregue nos próximos trabalhos. Um exemplo de documentação está no Moodle.

Obs2: informações sobre a forma de submissão do trabalho no moodle serão divulgadas posteriormente

Comentários Gerais:

1 Comece a fazer este trabalho logo, enquanto o problema está fresco na memória e o prazo para terminá-lo está tão longe quanto jamais poderá estar.

2 Clareza, indentação e comentários no programa também serão avaliados.

3 O trabalho é individual.

4 A submissão será feita pelo Moodle.

5 Trabalhos copiados serão penalizados conforme anunciado.

6 Penalização por atraso: $(2^d - 1)$ pontos, onde d é o número de dias de atraso.