BCC202 – Estruturas de Dados I (2022-01)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)
Estagiário docente: ——

Trabalho Prático 1 (TP1) - 10 pontos, peso 1.

- Submissão com data e hora de entrega disponíveis na plataforma da disciplina. O que vale é o horário do run.codes, e não do *seu*, ou do *meu* relógio!!!

- Clareza, identação e comentários no código também vão valer pontos. Por isso, escolha cuidadosamente o nome das variáveis e torne o código o mais legível possível.

- O padrão de entrada e saída deve ser respeitado exatamente como determinado no enunciado. Parte da correção é automática, não respeitar as instruções enunciadas pode acarretar em perda de pontos.

- Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.

- A avaliação considerará o tempo de execução e o percentual de respostas corretas.

- Eventualmente serão realizadas entrevistas sobre o trabalho para complementar a avaliação;

- O trabalho é em grupo de até 2 (duas) pessoas.

- Será aceito trabalhos após a data de entrega, todavia com um decréscimo de 0,05 a cada 10min.

- Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.

- Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.

- Códigos ou funções prontas específicas de algoritmos para solução dos problemas elencados não são aceitos

- Não serão considerados algoritmos parcialmente implementados.

- Procedimento para a entrega:.

1. Submissão: via **run.codes**.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via **run.codes**.
6. Você deve submeter os arquivos *.h*, *.c* e o *.pdf* (relatório) na raiz do arquivo *.zip*. Use os nomes dos arquivos *.h* e *.c* exatamente como pedido.
7. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.

- Bom trabalho!

Jogo da velha

O Jogo da velha é um jogo de regras extremamente simples. A estrutura do jogo é apenas um tabuleiro 3x3 posições, onde um jogador joga com o círculo (O) e outro com o **x** (X). Vence quem for o primeiro a fazer uma sequência de três símbolos iguais (trinca), seja em uma coluna, diagonal ou linha (ver Figura 1). Quando um jogador conquista o objetivo, costuma-se riscar a trinca vencedora. Quando há empate costuma-se dizer que o jogo "deu velha".

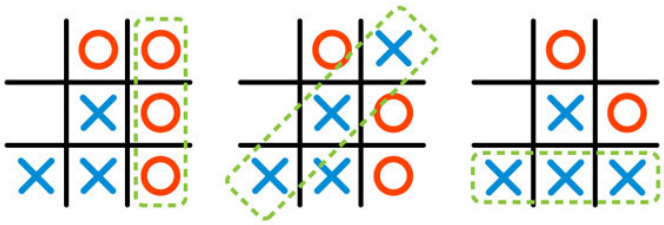

Figura 1: Situações de vitória no jogo da velha.

O seu objetivo é implementar um programa em linguagem C que permita ler tabuleiros de jogos da velha e assim indicar o cenário em que se encontra o jogo, seja ele inválido, andamento, vitória ou deu velha. A Figura 2 apresenta um fluxograma geral para o entendimento da proposta deste trabalho prático.

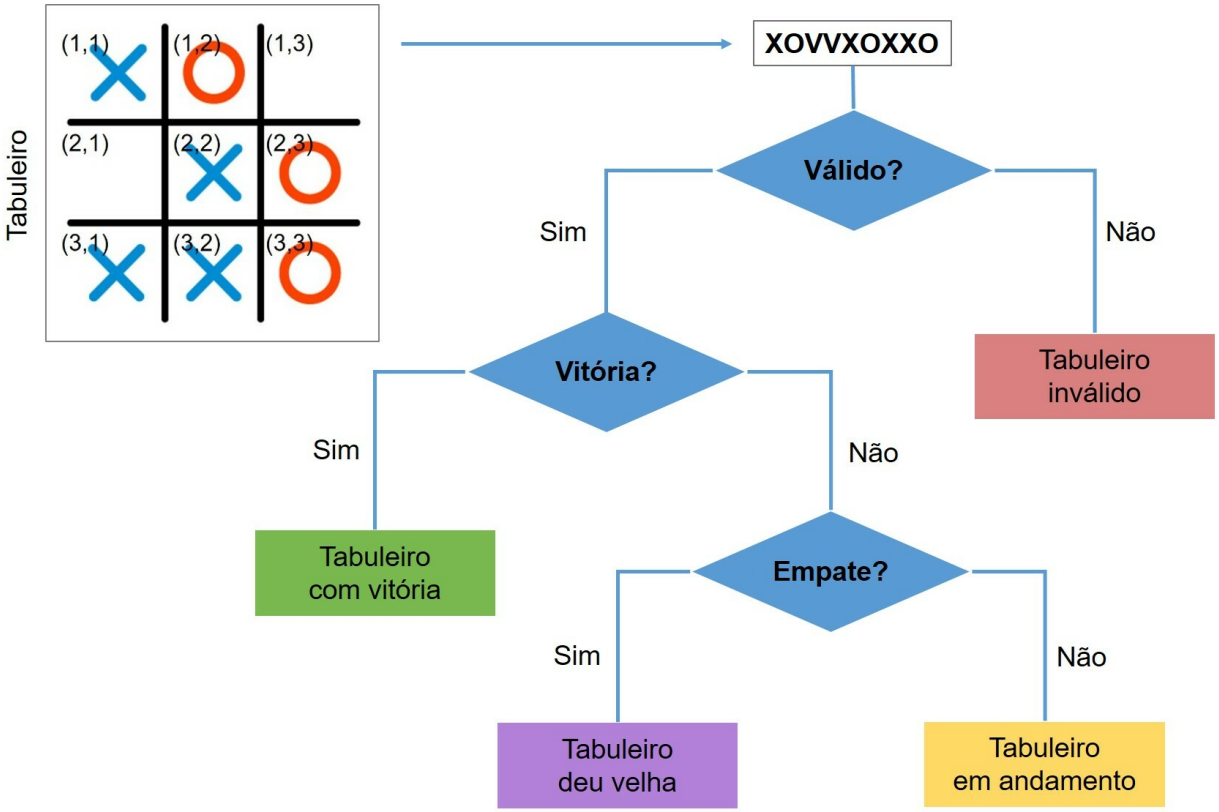

Figura 2: Fluxograma da análise de tabuleiros de jogo da velha.

Cada passo do fluxograma pode ser descrito como:

- **Leitura do tabuleiro**: Inicialmente é realizada a leitura do tabuleiro. A Figura 2 mostra um cenário típico de um tabuleiro em andamento e um mapeamento que será adotado para identificação das casas no tabuleiro. Observe que o tabuleiro é convertido em uma sequência de caracteres, sendo X a representação das jogadas do jogador 1; O a representação das jogadas do jogador 2 e V a representação das casas vazias, ou seja, as casas que poderão ser escolhidas em jogadas futuras. Na imagem, as casas (1,3) e (2,1) são aquelas que se encontram vazias.
- **Tabuleiro inválido**: A constatação de que o tabuleiro é válido permite descartar aqueles em condições inadequadas, como por exemplo, situações em que o tabuleiro apresenta jogadas desproporcionais para

um dos jogadores, ou ainda, a presença de mais de uma trinca vencedora para os casos de vitória.

- **Tabuleiro com vitória**: Os tabuleiros válidos podem configurar um cenário de vitória. Nesse caso, deve-se identificar o jogador vencedor. Observe que a vitória pode ocorrer sem que o tabuleiro esteja complemente preenchido. Tabuleiro deu velha: O caso de empate ou "deu velha" é aquele em que o jogo termina sem vencedor. Importante ressaltar que o tabuleiro não pode ter casas vazias (V), uma vez que há ainda possibilidade de vitória.
- **Tabuleiro em andamento**: Se o cenário não indicar vitória ou empate, então o jogo encontra-se em andamento. Nesse caso, é importante conferir se é possível identificar o jogador da vez e assim propor jogadas mestre de modo que ele consiga vencer o jogo. Lembre-se que apenas casas vazias podem ser sugeridas.

Assim, utilize suas habilidades de programação para implementar um tipo abstrato de dados (TAD) para o problema proposto e virar um *expert* no jogo da velha!

Imposições e comentários gerais

Neste trabalho, as seguintes regras devem ser seguidas:

- Seu programa não pode ter *memory leaks*, ou seja, toda memória alocada pelo seu código deve ser corretamente liberada antes do final da execução. (Dica: utilize a ferramenta *valgrind* para se certificar de que seu código libera toda a memória alocada).
- Um grande número de *Warnings* ocasionará a redução na nota final.

O que deve ser entregue

- Código fonte do programa em C (**bem identado e comentado**).
- Relatório do trabalho (relatório[1]). A documentação deve conter:
  1. **Introdução:** descrição sucinta do problema a ser resolvido e visão geral sobre o funcionamento do programa.
  2. **Implementação:** descrição sobre a implementação do programa. **Não faça** "*print screens*" de telas. Ao contrário, procure resumir ao máximo a documentação, fazendo referência ao que julgar mais relevante. É importante, no entanto, que seja descrito o funcionamento das principais funções e procedimentos utilizados, bem como decisões tomadas relativas aos casos e detalhes de especificação que porventura estejam omissos no enunciado. Muito importante: os códigos utilizados na implementação devem ser inseridos na documentação.
  3. **Testes:** descrição dos testes realizados e listagem da saída (não edite os resultados).
  4. **Análise**: deve ser feita uma análise dos resultados obtidos com este trabalho. Por exemplo, avaliar o tempo gasto de acordo com o tamanho do problema.
  5. **Conclusão**: comentários gerais sobre o trabalho e as principais dificuldades encontradas em sua implementação.
  6. **Bibliografia:** bibliografia utilizada para o desenvolvimento do trabalho, incluindo sites da Internet se for o caso.
  7. **Formato:** PDF ou HTML.

Como deve ser feita a entrega

Verifique se seu programa compila e executa na linha de comando antes de efetuar a entrega. Quando o resultado for correto, entregue via *Moodle*, um arquivo **.ZIP** com o nome e sobrenome do aluno. Esse arquivo deve conter: (i) os arquivos *.c* e *.h* utilizados na implementação, (ii) instruções de como compilar e executar o programa no terminal, e (iii) o relatório em **PDF**.

[1]Exemplo de relatório: `https://www.overleaf.com/latex/templates/modelo-relatorio/vprmcsdgmcgd`.

Detalhes da implementação

Para atingir seu objetivo, você deverá construir um Tipo Abstrato de Dados Tabuleiro como representação do Jogo da Velha que você quer analisar. O TAD deverá implementar, pelo menos, as seguintes operações:

1. `LeituraTabuleiro`: inicializa tabuleiro a partir de dados do terminal.
2. `TabuleiroEhValido`: retorna se o tabuleiro é válido ou não.
3. `Venceu`: verifica se alguém já venceu.
4. `Empate`: verifica se houve empate no jogo.
5. `JogadaMestre`: retorna as possíveis jogadas mestres disponíveis.

O TAD deve ser implementado utilizando a separação interface no *.h* e implementação *.c* discutida em sala, bem como as convenções de tradução. Caso a operação possa dar errado, devem ser definidos retornos com erro, tratados no corpo principal. A alocação da TAD necessariamente deve ser feita de forma dinâmica.

O código-fonte deve ser modularizado corretamente em três arquivos: *tp.c*, *jogo.h* e *jogo.c*. O arquivo *tp.c* deve apenas invocar e tratar as respostas das funções e procedimentos definidos no arquivo *jogo.h*. A separação das operações em funções e procedimentos está a cargo do aluno, porém, não deve haver acúmulo de operações dentro de uma mesma função/procedimento.

Você deve definir e implementar o `tp.c` (corpo principal - main - do seu programa) e outras operações para seu TAD. A implementação da Estrutura de Dados do TAD Tabuleiro deverá necessariamente utilizar um arranjo de duas dimensões alocado dinamicamente para representar o tabuleiro completo.

O limite de tempo para solução de cada caso de teste é de apenas **um segundo**. Utilize suas habilidades de programação e de análise de algoritmos para desenvolver um algoritmo correto e rápido!

Entrada

A entrada é dada por meio do terminal. Para facilitar, a entrada será fornecida por meio de arquivos. **Use eles como entrada no terminal com o seguinte comando:** `./executavel < arquivo_teste.in`. **NÃO É NECESSÁRIO LER O ARQUIVO.**

A entrada é dada por meio de um inteiro n e uma série de n de casos de teste. Cada linha de entrada equivale a um tabuleiro a ser analisado. As casas do tabuleiro são apresentadas em sequência com os caracteres **X** ou **O** para casas com jogadas já realizadas e **V** para casas vazias, como ilustrado na Figura 2.

Exemplo de entrada

```
7
OXVOXOXVO
OVVVXXVXO
VOXOOVOXX
OVVOOOOXO
OVXVVVXVO
XXOOOXXOO
XVVOXVOVX
```

Saída

Seu objetivo é construir um programa para verificar se um preenchimento (parcial ou completo) de um tabuleiro é válido e, caso afirmativo, verificar se há empate ou se alguém venceu e há uma (ou mais) jogada mestre. Deve-se informar ao usuário se o tabuleiro é inválido. O tabuleiro será lido pelo terminal.

A saída dependerá do estado do tabuleiro e deverá seguir exatamente os formatos sugeridos abaixo.

- Caso o tabuleiro seja inválido, o seu programa deve apresentar no terminal a seguinte mensagem: "`Tabuleiro invalido`".

- Caso o tabuleiro configure um cenário de vitória, além de informar que houve vitória, o jogador que atingiu o objetivo deve também ser informado entre colchetes, como no exemplo a seguir com vitória do jogador X: "`Tabuleiro com vitoria [X]`". Caso ocorra empate, a mensagem "`Tabuleiro deu velha`" deve ser apresentada.
- O cenário em andamento requer uma certa distinção nas mensagens. Além de informar "`Tabuleiro em andamento`", o seu programa deve especificar as seguintes condições entre colchetes, são elas:
  - "`Tabuleiro em andamento [proximo jogador indefinido]`" para os casos em que não seja possível identificar quem é o jogador da vez, ou seja, a quantidade de jogadas é a mesma para os dois jogadores, sendo impossível definir quem será o próximo.
  - "`Tabuleiro em andamento [X: sem jogada mestre]`" para os casos em que é possível definir o próximo jogador, porém não há jogada mestre para arrematar o jogo. Na mensagem anterior, o jogador X é o próximo a jogar, mas ainda não é possível vencer.
  - "`Tabuleiro em andamento [X: (2,1)(1,3)]`" quando se é possível identificar o próximo a jogar e suas respectivas jogadas mestre para ganhar. Considere informar as jogadas a partir da análise de **linha**, **coluna** e **diagonal** (principal e depois a secundária) de modo a evitar divergência com as mensagens dos casos de teste. Na mensagem apresentada, o jogador X pode ganhar o jogo da velha caso escolha as casas (2,1) ou (1,3) para o tabuleiro da Figura 2.

Exemplo de um caso de teste

Exemplo da saída esperada dada uma entrada:

Entrada	**Saída**

`7`
`OXVOXOXVO`
`OVVVXXVXO`
`VOXOOVOXX`
`OVVOOOOXO`
`OVXVVVXVO`
`XXOOOXXOO`
`XVVOXVOVX`
	`Tabuleiro 1 em andamento [X: (3,2)(1,3)]`
	`Tabuleiro 2 em andamento [O: sem jogada mestre]`
	`Tabuleiro 3 em andamento [X: (2,3)]`
	`Tabuleiro 4 invalido`
	`Tabuleiro 5 em andamento [proximo jogador indefinido]`
	`Tabuleiro 6 deu velha`
	`Tabuleiro 7 com vitoria [X]`

A SAÍDA DA SUA IMPLEMENTAÇÃO DEVE SEGUIR EXATAMENTE A SAÍDA PROPOSTA.

Diretivas de Compilação

```
$ gcc -c jogo.c -Wall
$ gcc -c tp.c -Wall
$ gcc jogo.o tp.o -o exe
```

PONTO EXTRA

Será concedido 0,1 extra para quem gerar o relatório em Latex (deve ser enviado os *.tex*).