**Universidade de São Paulo**
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências da Computação
SCC0201 — Introdução a Ciências da Computação II

# Trabalho 01: Campo Minado

**Professores**: Fernando Pereira dos Santos e Moacir Antonelli Ponti

**PAEs**: Edresson Casanova (C) e Leo Sampaio Ferraz Ribeiro (D)

**Monitores**: Gabriel Alves Kuabara (B) e Guilherme Amaral Hiromoto (A)

Desenvolva o trabalho sem olhar o de colegas.
Se precisar de ajuda pergunte, a equipe de apoio está aqui por você.

## 1 Objetivo do Trabalho

Você deverá implementar os diferentes estágios do clássico jogo "Campo Minado".

## 2 Campo Minado

Campo Minado é um popular jogo de computador para um jogador. Foi inventado por Robert Donner em 1989 e tem como objetivo revelar um campo de minas sem que alguma seja detonada. Este jogo tem sido reescrito para as mais diversas plataformas, sendo a sua versão mais popular a incluída nativamente com o Microsoft Windows.

### 2.1 Regras

A área de jogo consiste num campo de quadrados retangular. Cada quadrado pode ser revelado clicando sobre ele, e se o quadrado clicado contiver uma mina, então o jogo acaba. Se, por outro lado, o quadrado não contiver uma mina, uma de duas coisas poderá acontecer:

1. Um número aparece, indicando a quantidade de quadrados adjacentes que contêm minas;
2. Nenhum número aparece. Neste caso, o jogo revela automaticamente os quadrados que se encontram adjacentes ao quadrado vazio, já que não podem conter minas;

O jogo é ganho quando todos os quadrados que não têm minas são revelados.

# 3 Proposta

Seu programa deverá resolver as diversas etapas do jogo campo minado. Para facilitar o processo de desenvolvimento e correção, seu programa deverá implementar 3 diferentes módulos: (i) Leitura, (ii) inicialização do tabuleiro e (iii) ação do usuário. Cada um destes módulos deverão ser executados de acordo com uma opção inteira que será fornecida via stdin. Em seguida, ele deverá ler o nome do arquivo que contém o tabuleiro inicial do jogo. Deste modo, um exemplo de arquivo de entrada é:

```
1
1.board
```

Onde a opção 1 foi selecionada, com o tabuleiro descrito no arquivo `1.board`.

## 3.1 Tabuleiro

O arquivo que contém o tabuleiro é formado por caracteres '.' que indicam espaço vazio e '*' que indicam a presença de uma mina. As dimensões do tabuleiro podem mudar a cada caso, por isso use de alocação dinâmica e pela presença de quebra de linha `\n` ao final de cada linha. Um exemplo de um arquivo contendo um tabuleiro é:

```
....*
*....
.....
.*...
.....
```

## 3.2 Opção 1: Imprimir o tabuleiro

Caso a entrada indique a opção 1, seu programa deve apenas ler o tabuleiro e imprimi-lo na tela.

## 3.3 Opção 2: Imprimir o tabuleiro com as dicas

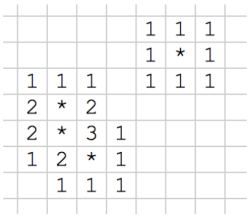


Figure 1: Tabuleiro preenchido com dicas numéricas

Seu programa deverá preencher o tabuleiro com as "dicas" numéricas. Essa dica é composta por um número que informa a quantidade de minhas em torno do espaço onde

está a dica (utilize como "vizinhança" os 8 espaços adjacentes, incluindo as diagonais). Esta dica numérica só deverá estar presente nos quadrantes que inicialmente estavam vazios e que possuem minas adjacentes. Um exemplo do preenchimento está na Figura 1.

É possível notar na figura como uma "borda" em torno das minhas é formada pelas dicas numéricas.

Após o processamento do tabuleiro com as dicas, seu programa deverá imprimir o resultado. Considerando o arquivo de exemplo, o resultado seria:

```
11.1*
*1.11
221..
1*1..
111..
```

## 3.4 Opção 3: Controle do Usuário

Quando essa opção é selecionada, o processamento do tabuleiro deve ser realizado como na opção 2, mas a impressão dependerá da ação tomada pelo usuário. Além da entrada usual, nesse caso uma linha extra de entrada deve ser lida, contendo a posição escolhida pelo usuário, como no exemplo:

```
3
1.board
0 4
```

As coordenadas dos quadrantes seguem a ordem de [linha, coluna]; no exemplo apresentado a posição selecionada foi linha 0, coluna 4.

Considerando a posição selecionada, existem três fluxos que seu programa pode seguir a depender do que existe naquela posição: (i) uma mina, (ii) uma dica ou (iii) nada.

### 3.4.1 Uma Mina Foi Selecionada

Se uma mina estiver na posição selecionada, imprima todo o tabuleiro da mesma forma como ocorre quando a opção 2 é usada. Isso deve indicar ao usuário que o jogo foi perdido. Esse seria o caso da posição [0, 4] do nosso exemplo, e a saída seria:

```
11.1*
*1.11
221..
1*1..
111..
```

### 3.4.2 Uma Dica Foi Selecionada

Se a posição selecionada conter uma dica, imprima o tabuleiro apenas com essa dica revelada, mantendo todas as outras posições secretas com o caractere 'X'. Caso a posição [0, 0] fosse selecionada em nosso exemplo, o resultado correto seria:

```
1XXXX
XXXXX
XXXXX
XXXXX
XXXXX
```

## 3.5 Um Espaço Vazio Foi Selecionado

Finalmente, o caso mais complicado acontece quando um espaço vazio é selecionado. Nesse caso, o tabuleiro deve revelar todos os espaços vazios, expandindo a "revelação" em todas as direções *até que uma borda ou uma dica seja encontrada*. Note que este é um cenário que naturalmente pede por uma solução recursiva e uma obrigatoriamente deve ser usada. Após o fim da "revelação", quando bordas e dicas pararem o processo, imprima o tabuleiro resultante. Na figura 2 mostramos um exemplo visual da expansão.

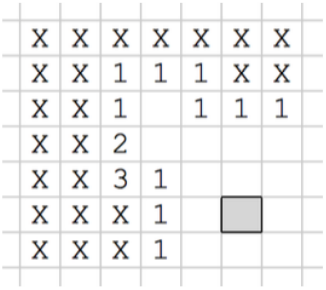


Figure 2: Tabuleiro após seleção de uma casa vazia revelar recursivamente todas as casas vazias em sua volta até que uma borda ou dica fosse encontrada. Note como os espaços com o caractere 'X' permanecem não revelados e podem ser minas, espaços vazios ou outras dicas.

Seguindo nosso exemplo com o `1.board`, caso a entrada informada seja [4, 4] (última linha, última coluna), o resultado que seu programa deverá imprimir é:

```
X1.1X
X1.11
X21..
XX1..
XX1..
```

Note como as posições reveladas foram expandidas nas 8 direções, incluindo as diagonais.

## 4 Submissão

Envie seu código fonte para o run.codes (apenas o arquivo `.c`).

1. **Crie um header com identifiação**. Use um header com o nome, número USP, código do curso e o título do trabalho. Uma penalidade na nota será aplicada se seu código estiver faltando o header.

2. **Comente seu código**. O objetivo é que tenhamos um código claro e que facilite a correção. Exemplos: Se uma variável deixa sua função clara em nome, um comentário sobre ela não é necessário; Um loop triplo que acessa um vetor de matrizes e altera os valores a depender da posição provavelmente pede por um comentário explicando a ideia por trás.

3. **Organize seu código em funções**. Use funções para deixar cada passo da execução mais clara, mesmo que a função seja chamada apenas uma vez. Comente sua função, descrevendo suas entradas e saídas, sempre que o nome da função e entradas não deixar isso óbvio.

4. **Utilize Alocação Dinâmica** para armazenar os elementos da matriz e não se esqueça de liberar a memória alocada ao fim da execução.

5. **Tire Dúvidas com a Equipe de Apoio**. Se não conseguiu chegar em uma solução, dê um tempo para descansar a cabeça e converse com a equipe de apoio sobre a dificuldade encontrada se precisar.