Trabalho de Introdução à Programação

Batalha Naval

Observações:

1. Trabalho deve ser feito em dupla;
2. O trabalho vale 5 pontos. 50% dos pontos se o trabalho for entregue no prazo e em perfeito funcionamento e os outros 50% condicionado a implementação/modificação do trabalho no dia da prova;
3. Linguagem: C.
4. Data de entrega: 16/04/2018.

1. Introdução

Batalha Naval é um jogo de tabuleiro de lápis e papel do qual participam dois jogadores. O objetivo é afundar a frota de navios do inimigo. Inicialmente é definido um tabuleiro (matriz) de 10 x 10, os tabuleiros não são visíveis entre os jogadores. Cada jogador usa dois tabuleiros. Antes de iniciar o jogo os jogadores posicionam a sua frota em seu primeiro tabuleiro, sem revelar ao adversário tal posicionamento. O segundo tabuleiro representa a área do adversário com todas as posições encobertas (isto é, sem informações sobre o que se encontra lá). As jogadas se dão de forma alternada.

Quando da sua vez, um jogador "atirar" em uma posição do tabuleiro do adversário indicando as suas coordenadas (linha e coluna). O jogo deve então informar se o tiro acertou algo ou não e, se o jogador acertou algo e todas as partes de uma embarcação já foram atingidas após o último "tiro", então tal embarcação foi afundada e o jogo deve informar o tipo de embarcação que afundou. Ao longo do jogo, o segundo tabuleiro registra a título de controle, o resultado de cada "tiro" que deu no tabuleiro do adversário identificando as partes das embarcações já atingidas. Ganha o jogo quem afundar primeiro toda a frota do adversário.

2. Objetivo

O principal objetivo do trabalho é praticar a programação em C, desenvolvendo uma versão preliminar de um jogo batalha naval. Em tal jogo, os usuários (dois) desempenham o papel de jogador e o computador o de mantenedor do "campo de batalha" (tabuleiro). Os usuários disparam "tiros" e o computador informa, na tela, o resultado de cada tiro no tabuleiro até que toda a frota de um jogador tenha sido afundada.

3. Implementação

O jogo deve lê o posicionamento inicial das embarcações de um arquivo texto. O estado inicial do tabuleiro é armazenado em tal arquivo, onde, cada linha representa uma linha do tabuleiro. O tabuleiro terá 10 linhas e 10 colunas enumeradas de 1 a 10. A quantidade e o formato de cada embarcação no tabuleiro são indicados na tabela abaixo:

Quantidade	Embarcação	Formato
1	Porta-aviões	PPPPP
1	Navio-tanque	TTTT
2	Contratorpedeiro	CCC
3	Submarino	SSS
4	Destroier	DD

O arquivo abaixo representa um exemplo de distribuição da frota em um tabuleiro. Para representar a água é utilizado o símbolo ~.

```
C~~~~~~~~S
C~~TTTT~~S
C~~~~~~~~S
~SSS~~DD~~
~~~~~S~~P~
~~~~~S~~PD
D~C~~S~~PD
D~C~~~~~P~
~~C~~~~~P~
SSS~~DD~~~
```

3.1. Carregamento dos arquivos

O programa dever ter uma função chamada carregarTabuleiros, que deve ler os arquivos dos jogadores e armazenar no tabuleiro de cada jogador (matriz). A função deve ter o seguinte protótipo:

```
int carregarTabuleiros(char arquivo1[], char arquivo2[],char tabuleiro1[][10], char tabuleiro2[][10]);
```

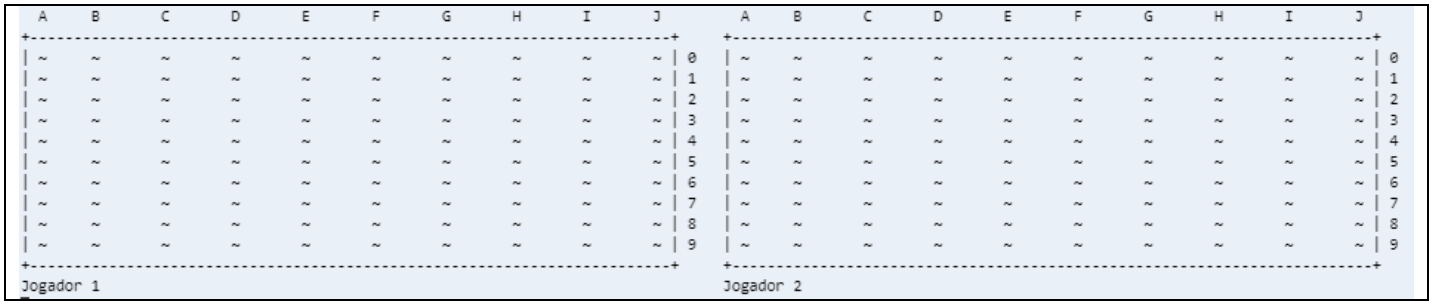
A função retorna 0 (zero) caso o carregamento dos arquivos sejam feito com sucesso e 1 (um) caso ocorra um erro de carregamento. Se houver um erro o programa deve parar a execução.

3.2. Imprimir os tabuleiros

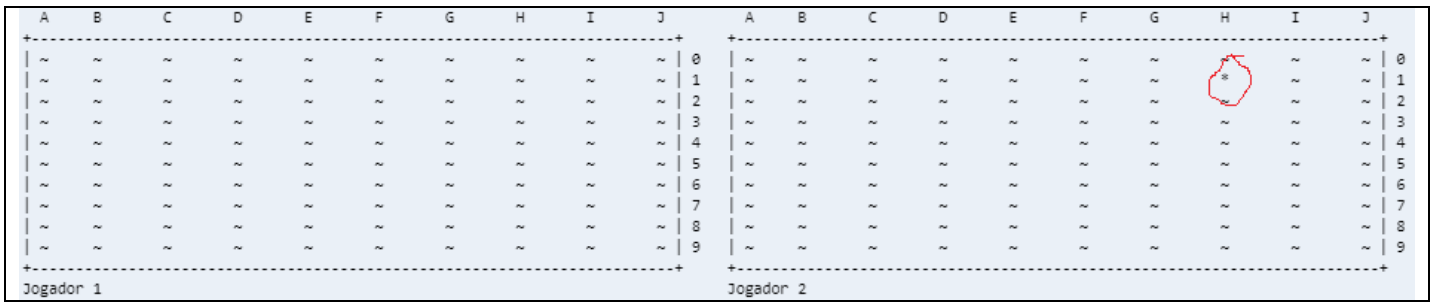
Para imprimir os tabuleiros na tela o programa deve chamar a função imprimirTabuleiros, que deve ler os tabuleiros (matriz), Lembre que as embarcações devem ficar oculta no tabuleiros, só deve ser apresentada as partes das embarcações já atingidas. A função deve ter o seguinte protótipo:

```
void imprimirTabuleiros(char tabuleiro1[][10], char tabuleiro2[][10]);
```

A função dever imprimir os tabuleiros como mostra a figura abaixo:



Quando uma parte da embarcação é atingida a representação da embarcação é substituída pelo símbolo * (fogo). A figura abaixo mostra um tabuleiro com a representação de parte de uma embarcação atingida.



Observação: Para limpar a tela em linguagem C, basta utilizar o seguinte comando: `printf("\e[H\e[2J")`.

3.3. Atirar

O jogo deve solicitar a cada jogador, na sua vez de jogar, que dispare 1 (um) "tiro", indicando as coordenadas do alvo através do número da linha e da letra da coluna que definem a posição. Para que o jogador tenha o controle dos "tiros" que acertam parte das embarcações, deverá marcar cada um deles na matriz do jogo (tabuleiro) com *. A função deve ter o seguinte protótipo:

```
int atirar (char tabuleiro[][10], int linha, int coluna);
```

A função deve retornar se o "tiro" acertou ou não parte de uma embarcação, retornando 0 (zero) quando acertar a água e 1 (um) quando acertar uma parte de uma embarcação. Quando acertar parte de uma embarcação o programa avisará que o "tiro" atingiu parte de uma embarcação. A função também retorna o valor 2 (dois) quando uma embarcação for totalmente destruída, se isso acontecer o programa informará qual embarcação foi afundada e as coordenadas dessa embarcação no tabuleiro será apagada, ou seja, o símbolo * (fogo) será substituído por ~ (água).

4. Execução

O programa (jogo) deve ser executado passando os seguintes parâmetros: (i) arquivo do jogador 1 e (ii) arquivo do jogador 2. Ou seja, na execução do programa devo passar através da linha de comando esses parâmetros. Veja um exemplo abaixo:

```
$ ./batalhaNaval jogador1.txt jogador2.txt
```

5. Entrega do código

O código e a documentação devem ser entregues em um arquivo compactado, contendo um arquivo readme.txt com o nome dos integrante e o comando para execução do código, e um arquivo PDF da documentação. Incluam todos os arquivos fontes (.c, .h, makefile, **não incluam executáveis ou arquivos objeto**). Um Makefile deve ser fornecido para a compilação do código.

Parte desse trabalho envolve o aprendizado de como construir um makefile e utilizar a ferramenta Make. Este makefile, qunaado executado sem parâmetros, irá gerar o programa batalhaNaval, EXATAMENTE com esse nome.

Entrega onde os programas não seguem as especificações de parâmetros e nomes, makefile não funcionando ou arquivos necessários faltando **não serão corrigidos**. Programas que não compilarem também não serão corrigidos.

6. Documentação

O texto da documentação deve ser breve, de forma que o corretor possa entender o que foi feito no código sem ter que entender linha a linha dos arquivos. Implementações modularizadas deverão mencionar quais funções são implementadas em cada módulo ou classe. A documentação deve conter os seguintes itens:

- Sumário do problema a ser tratado;
- Uma descrição sucinta dos algoritmos, das principais funções, e as decisões de implementação;
- Decisões de implementação que porventura estejam omissos na especificação;
- Testes, mostrando que o programa está funcionando de acordo com a especificação, seguidos da sua análise. Print screens mostrando o correto funcionamento do programa e exemplos de testes executados;
- Conclusões e referências bibliográficas.

7. Avaliação

A avaliação do trabalho será composta pela execução dos programas desenvolvidos e pela análise da documentação. Os seguintes itens serão avaliados:

- A qualidade do código (código bem organizado, com comentários explicativos, variáveis com nomes intuitivos, modularidade, etc).
- Execução correta do código em entradas de testes, a serem definidas no momento da avaliação. As entradas de teste irão exercitar a funcionalidade completa do código e testar casos especiais ou de maior dificuldade de implementação, mas que devem ser tratados por um programa correto.
- Conteúdo da documentação, que deve conter os itens mencionados anteriormente.
- Coerência e coesão da documentação (apresentação visual e organização, uso correto da linguagem, qualidade textual e facilidade de compreensão).

Como mencionado anteriormente, **trabalhos fora da especificação (por exemplo, sem makefile, que não gere o programa com o nome especificado, que não compile ou não possua os parâmetros esperados) não serão corrigidos**. Trabalhos fora da especificação tomam muito trabalho do corretor, que deve entender como compilar e executar cada programa avaliado, tempo esse que deveria ser empregado para avaliar a execução do código.

Casos de cópia não serão tolerados. Os códigos poderão ser comparados via ferramenta MOSS.

8. Avaliação

Os trabalhos poderão ser entregues até as 23:55 do dia especificado para a entrega. A fórmula para desconto por atraso na entrega do trabalho prático é:

$$\text{Desconto} = 2^{d-1}/0.32\%$$

Onde d é o atraso em dias. Note que após 5 dias, o trabalho não pode ser mais entregue.