



# Trabalho Prático II

---

## Regras Básicas

1. extends Trabalho Prático 01
2. **Fique atento ao Charset dos arquivos de entrada e saída.**

## Classe + Registro



Steam é uma plataforma criada em 2003 pela Valve que revolucionou o mundo dos games. O serviço apresentou recursos inovadores como o download de jogos em formato digital, o desbloqueio de conquistas (achievements) e a interação entre usuários. Atualmente, milhões de pessoas utilizam o serviço todos os dias. O download do programa é de graça, mas muitos games são pagos, embora a Steam ocasionalmente libere alguns jogos grátis durante finais de semana.

O serviço possui hoje aplicativos para Android e iOS e versões para Mac e Linux. Além disso, a versão do Brasil conta com preços em reais, e você pode pedir reembolso de jogos que não rodaram ou que não tenham atendido às suas expectativas.

Neste Trabalho Prático sua tarefa é organizar as informações dos jogos disponíveis para exibição ao usuário. Entretanto, esses dados estão espalhados em um arquivo `.csv` obtidos a partir de uma busca na plataforma. Todos esses dados estão agrupados no arquivo `games.csv`, e o mesmo deve ser colocado na pasta `/tmp/`.<sup>1</sup> Para isso, você deve ler, organizar e armazenar os dados de cada jogo em memória, utilizando as estruturas de dados em aula (Lista, Pilhas, Filas, Árvores e Hash). Em seguida executar as operações descritas nos arquivos de entrada. Muito cuidado ao realizar o *parser* do texto. Fique atento a descrição dos dados que serão lidos e manipulados pelo seu sistema.

---

<sup>1</sup>Quando reiniciamos o Linux, ele normalmente apaga os arquivos existentes na pasta `/tmp/`.

1. **Classe Game em Java:** Crie uma classe *Game* seguindo todas as regras apresentadas no slide unidade001\_nivelamento\_introducaoOO.pdf. Sua classe terá os atributos privados *app\_id* (Inteiro), *name* (String), *release\_date* (Data), *owners* (String), *age* (Inteiro), *price* (Float), *dlcs* (Inteiro), *languages* (Vetor de Strings), *website* (String), *windows* (Boolean), *mac* (Boolean), *linux* (Boolean), *upvotes* (Float), *avg\_pt* (Inteiro), *developers* (String), *genres* (Vetor de Strings). Ela terá também pelo menos dois construtores, e os métodos *gets*, *sets*, *clone* e *imprimir* e *ler*. O método *imprimir* mostra a String “*app\_id name release\_date owners age price dlcs [languages] website windows mac linux upvotes avg\_pt developers [genres]*”, contendo todos os atributos da classe. O método *ler* deve efetuar a leitura dos atributos de um registro. Veja que os dados estão divididos em vários arquivos.

A entrada padrão é composta por várias linhas e cada uma contém o identificador de um jogo. A última linha da entrada contém *FIM*. A saída padrão também contém várias linhas, uma para cada registro contido na entrada padrão, conforme o exemplo abaixo:

```
730 Counter-Strike: Global Offensive Aug/2012 50000000 - 100000000 0 0.00 1
[Czech, Danish, Dutch, English, Finnish, French, German, Hungarian, Italian,
Japanese, Korean, Norwegian, Polish, Portuguese, Portuguese - Brazil, Romanian,
Russian, Simplified Chinese, Spanish - Spain, Swedish, Thai, Traditional
Chinese, Turkish, Bulgarian, Ukrainian, Greek, Spanish - Latin America,
Vietnamese] http://blog.counter-strike.net/ true true true 88% 508h 4m
Valve,Hidden Path Entertainment [Action, Free to Play]
```

2. **Registro em C:** Repita a anterior criando o registro *Game* na linguagem C.

## Pesquisa

3. **Pesquisa Sequencial em Java:** Faça a inserção de alguns objetos no final de uma Lista e, em seguida, faça algumas pesquisas sequenciais. A chave primária de pesquisa será o atributo **nome do game**. A entrada padrão é composta por duas partes onde a primeira é igual a entrada da primeira questão 1. As demais linhas correspondem a segunda parte. A segunda parte é composta por várias linhas. Cada uma possui um elemento que deve ser pesquisado na Lista. A última linha terá a palavra *FIM*. A saída padrão será composta por várias linhas contendo as palavras *SIM/NAO* para indicar se existe cada um dos elementos pesquisados. Além disso, crie um arquivo de log na pasta corrente com o nome *matrícula\_sequencial.txt* com uma única linha contendo sua matrícula, tempo de execução do seu algoritmo e número de comparações. Todas as informações do arquivo de log devem ser separadas por uma tabulação ‘\t’.

4. **Pesquisa Binária em Java:** Repita a questão anterior, contudo, usando a Pesquisa Binária. A entrada e a saída padrão serão iguais às da questão anterior. O nome do arquivo de log será matrícula\_binaria.txt.

## Estruturas Sequenciais

5. **Lista com Alocação Sequencial em Java:** Crie uma Lista de Games baseada na lista de inteiros vista na sala de aula. Sua lista deve conter todos os atributos e métodos existentes na lista de inteiros, contudo, adaptados para a classe *Game*. De toda forma, lembre-se que, na verdade, temos uma lista de ponteiros e cada um deles aponta para um objeto *Game*. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa lista.

Os métodos de inserir e remover devem operar conforme descrito a seguir, respeitando parâmetros e retornos. Primeiro, o *void inserirInicio(Game game)* insere um objeto na primeira posição da Lista e remaneja os demais. Segundo, o *void inserir(Game game, int posição)* insere um objeto na posição *p* da Lista, onde *p < n* e *n* é o número de objetos cadastrados. Em seguida, esse método remaneja os demais objetos. O *void inserirFim(Game game)* insere um objeto na última posição da Lista. O *Game removerInicio()* remove e retorna o primeiro objeto cadastrado na Lista e remaneja os demais. O *Game remover(int posição)* remove e retorna o objeto cadastrado na *p*-ésima posição da Lista e remaneja os demais. O *Game removerFim()* remove e retorna o último objeto cadastrado na Lista.

A entrada padrão é composta por duas partes. A primeira é igual a entrada da primeira questão. As demais linhas correspondem a segunda parte. A primeira linha da segunda parte tem um número inteiro *n* indicando a quantidade de objetos a serem inseridos/removidos. Nas próximas *n* linhas, tem-se *n* comandos de inserção/remoção a serem processados neste exercício. Cada uma dessas linhas tem uma palavra de comando: II inserir no início, I\* inserir em qualquer posição, IF inserir no fim, RI remover no início, R\* remover em qualquer posição e RF remover no fim. No caso dos comandos de inserir, temos também o *app\_id* do registro a ser inserido. No caso dos comandos de “em qualquer posição”, temos também a posição. No Inserir, a posição fica imediatamente após a palavra de comando. A saída padrão tem uma linha para cada objeto removido sendo que essa informação será constituída pela palavra “(R)” e o atributo **nome**. No final, a saída mostra os atributos relativos a cada objeto cadastrado na lista após as operações de inserção e remoção.

6. **Pilha com Alocação Sequencial em Java:** Crie uma Pilha de Games baseada na pilha de inteiros vista na sala de aula. Neste exercício, faremos inserções, remoções e mostraremos os elementos de nossa pilha. A entrada e a saída padrão serão como as da questão anterior, contudo, teremos apenas os comandos I para inserir na pilha (empilhar) e R para remover (desempilhar).

**7. Fila Circular com Alocação Sequencial em Java:** Crie uma classe *Fila Circular* de *Game*.

Essa fila deve ter tamanho cinco. Em seguida, faça um programa que leia vários registros e insira seus atributos na fila. Quando o programa tiver que inserir um objeto e a fila estiver cheia, antes, ele deve fazer uma remoção. A entrada padrão será igual à da questão anterior. A saída padrão será um número inteiro para cada registro inserido na fila. Esse número corresponde à média **arredondada** do atributo `avg.pt` dos registros contidos na fila após cada inserção. O final da saída padrão mostra os registros existentes na fila seguindo o padrão da questão anterior.

**8. Lista com Alocação Sequencial em C:** Refaça a questão 5 na linguagem C.