

U.C. 21077

Linguagens de Programação e-Fólio A – Linguagem OCaml e Java

-- INSTRUÇÕES --

- 1)** O e-fólio tem uma cotação de 4 valores.
- 2)** Qualquer tentativa de plágio resultará numa nota final de zero valores.
- 3)** Este e-fólio deve ser resolvido usando a *linguagem Java e OCaml*.
- 4)** Deve ser submetido um ficheiro comprimido (ZIP ou RAR) com o nome e número de estudante contendo:
 - a) Código do programa devidamente comentado;
 - b) **Ficheiro readme.txt** com a informação necessária para compilar e executar o programa;
 - c) **Relatório em formato .pdf até 4 páginas** descrevendo a solução apresentada e os testes efetuados

E-fólio A

O Sr. Ganâncio, dono da oficina "Mecânica do Ganâncio", está cansado de fazer orçamentos manualmente e decidiu contratá-lo, um talentoso estudante de programação, para desenvolver um sistema que automatize esse processo.

A oficina possui:

- Um inventário de peças (óleos, velas, baterias, etc.) com custos, preços de venda e quantidades.
- Serviços mecânicos que exigem certas categorias de peças e tempo de mão de obra.
- Mecânicos com custos por hora diferentes.
- Descontos aplicáveis a marcas específicas e ao tempo de serviço.

O Sr. Ganâncio quer que o sistema:

- Selecione as peças mais lucrativas para cada serviço (maior diferença entre preço com desconto e custo).
- Calcule custos de mão de obra, considerando descontos por tempo de serviço.
- Gere orçamentos finais combinando peças, serviços e descontos.

O Sr. Ganâncio entregou-lhe uma base de dados feita por um amigo que lhe disse entender “da coisa” mas depois desistiu, o ficheiro “*database.pl*”.

Regras imprescindíveis do Sr. Ganâncio:

- OCaml: Processamento principal (leitura, cálculos, filtros).
- Java: Interface e consolidação (apenas exibe e consolida resultados).
 - Apesar do Java ser apenas uma Interface, as boas práticas de Programação por Objetos devem ser aplicadas, bem como o bom uso de Classes e modularidade, a aplicação deve ser escalável facilmente modificável e com verificação de Erros.
- Código limpo: Comentários, nomes descritivos e tratamento de erros.

Dica: Use separadores consistentes (ex.: ;) nas saídas OCaml para facilitar a leitura no Java. *Leia atentamente todas as questões primeiro* para garantir que pensa sempre na integração Java-Ocaml.

Boa sorte! O Sr. Ganâncio está ansioso pelo teu sistema revolucionário! 🚗 🎉

1. Leitura e Organização da Base de Dados (OCaml) - 0,5 Valores

O ficheiro *database.pl* contém dados em formato Prolog.

Implemente funções para:

- a) Ler a base de dados (*para o ajudar veja o exemplo abaixo de como se lê e recolhe os itens em inventário ex.: ./main.exe listar_items*)
- b) Ordenar itens por categoria → marca → nome e retorná-los formatados (*ex.: ID; Nome; Marca; Tipo; Custo; Preço; Quantidade*). Estamos a falar de uma lista só que é ordenada por ordem A→Z para as seguintes variáveis categoria → marca → nome por essa ordem.

```

let read_file filename =
  let lines = ref [] in
  let channel = open_in filename in
  try
    while true do
      lines := input_line channel :: !lines
    done; []
  with End_of_file ->
    close_in channel;
    List.rev !lines

let parse_item line =
  let regexp = Str.regexp "item(\\"([0-9]+\\\"), \\"([^\"]+\\\"), \\"([^\"]+\\\"), \\"([^\"]+\\\"), \\"([0-9.]+\\\"), \\"([0-9.]+\\\"), \\"([0-9.]+\\\"), \\"([0-9.]+\\\"))" in
  if Str.string_match regexp line 0 then
    Some (
      int_of_string (Str.matched_group 1 line), (* ID *)
      Str.matched_group 2 line, (* Nome *)
      Str.matched_group 3 line, (* Marca *)
      Str.matched_group 4 line, (* Tipo *)
      float_of_string (Str.matched_group 5 line), (* Custo *)
      float_of_string (Str.matched_group 6 line), (* Preço Venda *)
      int_of_string (Str.matched_group 7 line) (* Quantidade *)
    )
  else
    None

let () =
  match Array.to_list Sys.argv with
  | _ :: "listar_items" :: _ ->
    let file_lines = read_file "/home/rudigualter/projects/e-folioA/database/database.pl" in
    let itens = List.filter_map parse_item file_lines in
    print_items itens
  | _ ->
    Printf.printf "Comandos Disponíveis:\n";
    Printf.printf " listar_items\n";

```

2. Seleção de Peças para Serviços (OCaml) – 1,5 Valores

Dada uma lista de IDs de serviços (ex.: ./main.exe orcamento_items 1,2), retorne as peças mais lucrativas para cada categoria necessária desse(s) serviços.

Dica: *Lucro = (Preço com desconto da marca) - Custo. Para Cada Categoria no Serviço escolhemos a peça mais rentável, devolvemos a lista de peças para os serviços pretendidos.*

3. Cálculo de Mão de Obra (OCaml) – 0,5 Valores

Dada uma lista de IDs de serviços, para cada serviço, calcule, horas trabalhadas, custo hora, custo sem desconto, desconto (*se tempo < 0.25h ou > 4h*), e custo total.

Ex.: ./main.exe orcamento_mecanico 1,5 (Saída: ID Serviço; Horas; Custo Hora; Custo Sem Desconto; Desconto; Total)

Crie uma classe Cart com atributos apropriados. A classe Cart representará um carrinho de compras e terá atributos apropriados para armazenar os itens adicionados pelo cliente. Crie um carrinho fictício para o cliente, para os seus testes.

4. Descontos em Peças e Custo Fixo (OCaml) 0,5

a) Para as peças selecionadas dos serviços, liste o desconto aplicado (por marca) e o preço final.

Ex.: ./main.exe orcamento_desconto_items 1,5 (Saída: ID Item; Desconto (%); Valor do Desconto)

b) Para os serviços obtenha o preço fixo dos mesmos.

Ex.: ./main.exe orcamento_preco_fixo 1,5 (Saída: ID Serviço; Valor)

5. Sistema de Orçamento (Java) – 1 Valor

Use Java para chamar os executáveis OCaml via ProcessBuilder e exibir resultados.

Requisitos:

- Classes bem organizadas (ex.: Oficina, Orcamento, IntegradorOCaml).
- Leia as saídas do OCaml e gere um orçamento final consolidando:
 - Preço total das peças.
 - Custo total da mão de obra.
 - Custos Fixos
 - Descontos aplicados.
 - Total

BONUS: 0.1 Valores para quem fizer os cálculos do orçamento final também em Ocaml, só aplicado quando a nota final é inferior a 4 valores.

Notas:

1. (C3) Todas as escolhas devem ser fundamentadas no relatório.
2. (C1) A forma de implementar os vários módulos propostos.
3. (C2) A forma e lógica do programa fica ao critério de cada estudante, mantendo as boas práticas de programação.
4. (C2) A facilidade de utilização do programa é valorizada (exemplo: menus de acesso, e outras estruturas e termos complexos)