UNIVERSIDADE ABERTA
www.uab.pt

Ministério da Educação e Ciência

e-Fólio

U.C. 21077

Linguagens de Programação

e-Fólio A – Linguagem OCaml

-- 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 OCaml*.

4) Deve ser submetido um ficheiro comprimido (ZIP ou RAR) com o nome e número de estudante contendo:

a) Código do programa;

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

Neste trabalho, é proposto que cada estudante implemente uma **estrutura de pilha** usando OCaml que simule um exercício de Logística. Uma pilha é uma estrutura de dados *last-in*, *first-out* (LIFO) que permite empilhar elementos no topo da pilha e desempilhar elementos do topo da pilha.

O contexto de aplicação é seguinte: você acabou de ser contratado para uma empresa de logística que precisa controlar os pacotes que estão a ser carregados num camião. Os pacotes são carregados no camião numa ordem específica e são descarregados na ordem inversa. Para controlar os pacotes, um funcionário decidiu implementar um módulo em OCaml que usava uma estrutura de dados de pilha, entretanto esse funcionário foi de férias e deixou a sua implementação incompleta, então a tarefa de continuação foi atribuída a você. Os itens são carregados no camião sempre que um item é carregado o id do item é registado. Os Ids são números inteiros positivos.

O Código Criado pelo seu colega encontra-se em anexo no ficheiro *logi.ml*.

1. Implementando a Pilha

Nesta parte da tarefa, você estará implementando o módulo Pilha (***STACK***) usando o tipo de módulo já criado pelo seu colega que foi de férias. Com as seguintes funções:

empty : cria uma pilha vazia.

is_empty : retorna ***true*** se a pilha estiver vazia, ***false*** caso contrário.

push : adiciona um elemento ao topo da pilha.

peek : retorna o elemento do topo da pilha sem removê-lo.

pop : remove o elemento do topo da pilha e o retorna.

size : retorna o número de elementos na pilha.

O seu colega informou-o que a implementação já estava quase terminada apenas precisava de uma simples linha.

2. Implementando o Testador de Pilha

Nesta parte da tarefa, você estará implementando o módulo Testador de Pilha (***StackTester***). O módulo Testador de Pilha fornece funções para construir e testar pilhas. Você deve fornecer as seguintes funções:

build : recebe um parâmetro de construção e retorna uma pilha construída a partir das instruções no parâmetro de construção.

test : recebe uma pilha e um parâmetro de verificação e retorna ***true*** se a pilha satisfaz as condições no parâmetro de verificação, ***false*** caso contrário.

3. Implementando a Funcionalidade de Impressão de Pilha

Nesta parte da tarefa, você estará implementando a função imprimir no módulo Testador de Pilha (***StackTester***).

print : A função imprimir recebe uma função que converte um elemento numa string e uma pilha e imprime os elementos da pilha.

4. Testando sua Implementação

Nesta parte da tarefa, você estará testando a sua implementação do módulo Pilha (***STACK***) e do módulo Testador de Pilha (***StackTester***). Você deve fornecer pelo menos 3 testes seus e realizar os seguintes 2 testes:

- O camião chegou no centro logístico e carregou os IDs pela seguinte ordem 5,6,8, ele realizou duas paragens durante o dia onde descarregou um item, qual o ID do item que ficou no camião?
- O camião chegou no centro logístico e carregou os IDs pela seguinte ordem 2,1,5,9,10, na primeira paragem descarregou dois itens e carregou um item com o ID 8, na paragem seguinte descarregou 2 itens, na paragem seguinte carregou um item com o ID 3. No final do dia que itens ficarão no camião?

O seu colega deixou um e-mail para o direcionar de como realizar o teste:

##############

Caro colega,

Você precisa implementar seu próprio módulo de pilha (por exemplo, MyStack) para testar o módulo *StackTester*. Isso ocorre porque o módulo *StackTester* recebe um módulo *Stack* como argumento, e o objetivo dos testes é garantir que o módulo StackTester funcione corretamente com diferentes implementações de pilha.

Olhe esta estrutura:

```
module MyStack : STACK = struct
(* implementamos o módulo de acordo com o STACK, basicamente instanciar as variáveis*)
End
```

Agora criamos um módulo que usa a nossa estrutura da nossa pilha.

```
module MyStackTester = StackTester(MyStack)
```

Assim podemos testar uma operação, por exemplo colocar os ID 1,2 e 3 e depois retiramos dois dos Ids:

```
let test = Push (1, Push (2, Push (3, Pop (Pop Empty))))
let result = MyStackTester.build test
let () = MyStackTester.print……..
```

Bom trabalho

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 de apresentar uma pilha fica ao critério de cada estudante.
4. (C2) A facilidade de utilização do programa é valorizada (exemplo: menus de acesso, e outras estruturas e termos complexos)