Lista de Exercícios – Estrutura de Dados

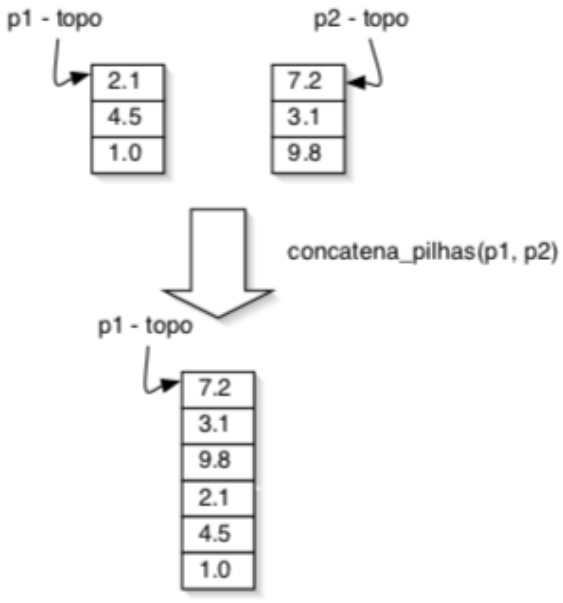
1) Implemente uma pilha e suas principais funções (cria, push, pop, vazia, libera) utilizando uma lista encadeada ao invés de um vetor na estrutura da pilha.

2) Considere a existência de um tipo abstrato Pilha de números de ponto flutuante, cuja interface está definida no arquivo pilha.h da seguinte forma:

```
typedef struct pilha Pilha;
Pilha* cria(void);
void push (Pilha* p, float v);
float pop (Pilha* p);
int vazia (Pilha* p);
void libera (Pilha* p);
```

a) Sem conhecer a representação interna desse tipo abstrato Pilha e usando apenas as funções declaradas no arquivo pilha.h, implemente uma função que receba uma pilha como parâmetro e retorne o valor armazenado em seu topo, restaurando o conteúdo da pilha.

b) Sem conhecer a representação interna desse tipo abstrato Pilha e usando apenas as funções declaradas no arquivo pilha.h, implemente uma função que receba duas pilhas, p1 e p2, e passe todos os elementos da pilha p2 para o topo da pilha p1. A figura a seguir ilustra essa concatenação de pilhas:



Note que ao final dessa função, a pilha p2 vai estar vazia e a pilha p1 conterá todos os elementos das duas pilhas. Essa função deve obedecer o protótipo:

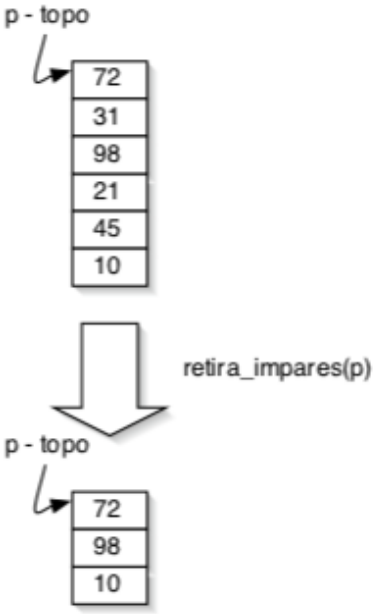
void concatena_pilhas (Pilha* p1, Pilha* p2);

Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar para fazer a transferência dos elementos entre as duas pilhas.

2) Considere a existência de um tipo abstrato Pilha de números inteiros, cuja interface está definida no arquivo pilha.h da seguinte forma:

```
typedef struct pilha Pilha;
Pilha* cria(void);
void push (Pilha* p, int v);
int pop (Pilha* p);
int vazia (Pilha* p);
void libera (Pilha* p);
```

Sem conhecer a representação interna desse tipo abstrato Pilha e usando apenas as funções declaradas no arquivo pilha.h, implemente uma função que receba uma pilha e retire todos os elementos impares dessa pilha. A figura a seguir ilustra o resultado dessa função sobre uma pilha:



Essa função deve obedecer o protótipo:

void retira_impares (Pilha* p);

Dica: Essa função pode ser implementada mais facilmente através de uma solução recursiva ou utilizando uma outra variável pilha auxiliar.

4) Implemente uma fila e suas principais funções (cria, insere, retira, vazia, libera) utilizando uma lista encadeada ao invés de um vetor na estrutura da fila.

5) Crie uma função que receba um vetor v de inteiros e retorne esse vetor com os valores ordenados. Crie este algoritmo da ordenação usando apenas a sua criatividade, sem pesquisar em livros, internet ou IAs.

6) Crie uma segunda função que receba um vetor v de inteiros e retorne esse vetor com os valores ordenados. Desta vez você deve implementar uma estratégia de ordenação diferente da que foi utilizada na questão anterior (ex: quicksort, heapsort, mergesort...) e pode pesquisar em livros, internet ou IAs.

7) Crie uma função que gere dois vetores inteiros de 1 milhão de posições cada. O primeiro vetor deve ser inicializado com os seus valores ordenados e o segundo com valores desordenados. Chame as funções das questões 5 e 6 para ordenar ambos os vetores e verifique o tempo de execução de cada algoritmo de ordenação. Existe diferença no tempo de execução ao compararmos os algoritmos de ordenação das questões 5 e 6? Qual é o mais rápido? Existe diferença no tempo de execução do vetor ordenado comparado ao vetor desordenado? OBS.: Caso necessário, utilize um vetor maior que 1 milhão de posições.