

Pilhas e Filas

Estrutura de Dados

Prof. Roberto Cabral

21 de Novembro de 2017

1. Suponha que um dado problema requer o uso de duas pilhas, onde cada pilha suporta no máximo 50 elementos e em nenhum momento as duas pilhas terão juntas mais do que 80 elementos. Assim, é possível implementar as duas pilhas em um único vetor usando apenas 80 posições ao invés de 100. Implemente a estrutura de dados e as de empilhar e desempilhar para estas duas pilhas.
2. Escreva um programa que implemente uma fila circular utilizando uma lista linear. O programa deve ser capaz de inserir, remover e informar o tamanho da fila em um dado momento.
3. Dada uma fila de inteiros, escreva um programa que exclua todos os números negativos sem alterar a posição dos outros elementos da fila.
4. Considere uma pilha p não vazia e uma fila f vazia. Utilizando apenas os testes de fila e pilha vazias, as operações `fila_insere`, `fila_retira`, `pop`, `push`, e uma variável `aux` do tipo da pilha, escreva uma função que inverta a ordem dos elementos da pilha.
5. Utilizando um dos TADs vistos em sala de aula (Lista, Pilha ou Fila) para auxiliá-lo na manipulação dos dados, implemente uma função que compute a fatoração prima de um número imprimindo os seus fatores em ordem crescente. Por exemplo, para o número 630 deverá ser impresso $2 \times 3 \times 3 \times 5 \times 7$. Justifique a escolha do TAD utilizado.
6. Considere o problema de verificar se uma sequência de parênteses e colchetes está balanceada. Por exemplo: “()”, “[()]”, “((([]])))” e “[]” são exemplos de sequências balanceadas enquanto “(”, “[]”, “[(]” e “[[()]” não são. Escreva uma função que recebe uma sequência de parênteses e colchetes dada por uma string e que devolve 1 caso a sequência esteja balanceada ou 0 caso contrário.
Obs.: Caso você queira usar um dos tipos abstratos de dados estudados em sala, você pode definir qual o tipo está usando e quais funções estão no seu “tipo.h” e se as funções já foram implementadas em alguma aula, elas podem ser usadas na construção da sua solução sem a necessidade de reimplementá-las.
7. Escreva uma função iterativa que simule o comportamento da seguinte função recursiva. Use uma pilha.

```

1  int ttt (int *x, int n){
2      if(n==-1) return 0;
3      if(x[n] > 0) return x[n] + ttt(x,n-1);
4      else return ttt(x,n-1);
5  }

```

8. Faça uma função que receba como entrada duas pilhas p_1 e p_2 e retorna 1 se as pilhas forem iguais e 0 caso contrario.
9. Considere uma pilha p vazia e uma fila f não vazia. Utilizando apenas os testes de fila e pilha vazias, as operações **fila_insere**, **fila_retira**, **pop**, **push**, e uma variável **aux** do tipo da fila, escreva uma função que inverta a ordem dos elementos da fila.
10. Para um dado número inteiro $n > 1$, o menor inteiro $d > 1$ que divide n é chamado de fator primo. é possível determinar a fatora  o prima de n achando-se o fator primo d e substituindo n pelo quociente n/d , repetindo essa opera  o at   que n seja igual a 1. Utilizando um dos TADs vistos em sala (Lista, Pilha ou Fila) para auxili  -lo na manipula  o dos dados, implemente uma fun  o que compute a fatora  o prima de um n  mero imprimindo os seus fatores em ordem decrescente. Por exemplo, para $n=3960$, dever   ser impresso $11 * 5 * 3 * 2 * 2 * 2$. Justifique a esquelha do TAD utilizado.
11. Fa  a uma fun  o **pop** alternativa que recebe como entrada um par  metro n e desempilha n elementos da pilha. A fun  o deve retornar um vetor com os elementos removidos.
12. Usando apenas as fun  es **push** e **pop**, implemente uma fun  o que receba uma pilha p como entrada e retorna a c  pia dessa pilha.
13. Fa  a um programa que recebe como entrada um n  mero inteiro e retorna seu respectivo valor em bin  rio usando Pilha.
14. Como voc   implementaria uma fila de pilhas? Uma pilha de filas? Uma fila de filas? Escreva rotinas para implementar as opera  es corretas para cada uma destas estruturas de dados.
15. Implemente uma fila onde cada item da fila consista em um n  mero vari  vel de inteiros.
16. Implemente o **quicksort** iterativo.
17. No estoque de uma grande empresa todas as caixas possuem pesos: 13, 11 e 7 toneladas. H   tr  s pilhas p_1 , p_2 e p_3 . Na pilha p_1 encontram-se todas as caixas que chegam no dep  sito. Com um detalhe: caixas maiores n  o podem ser empilhadas sobre caixas menores. Implemente uma fun  o chamada **chegada**(Caixa $*nova$, Pilha $*p$) que efetue o controle das caixas, de forma que caso uma caixa de maior peso do que uma que j   est   em p_1 deva ser empilhada, ent  o, todas as caixas que est  o em p_1 s  o movidas

para as pilhas auxiliares p_2 (contendo somente caixa de 11 toneladas) e p_3 (contendo somente caixas de 7 toneladas) até que se possa empilhar a nova caixa. Depois, todas as caixas são movidas de volta para a pilha p_1 . Crie e utilize as funções com os seguintes protótipos:

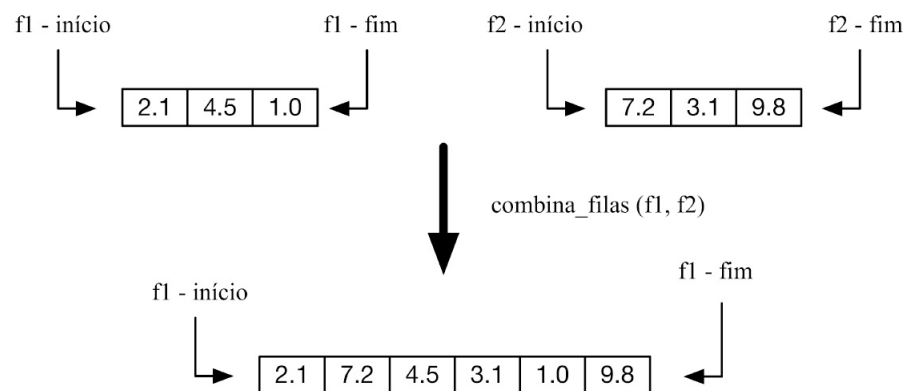
```
int vazia(Pilha *p);
int cheia(Pilha *p);
void empilhar(Caixa *nova, Pilha *p);
void desempilhar(Pilha *p);
Caixa* topo(tPilha *p);
```

Obs.: uma caixa deve conter seu peso e descrição.

18. Considere a existência de um tipo abstrato Fila de números reais, cuja interface está definida no arquivo “fila.h” da seguinte forma:

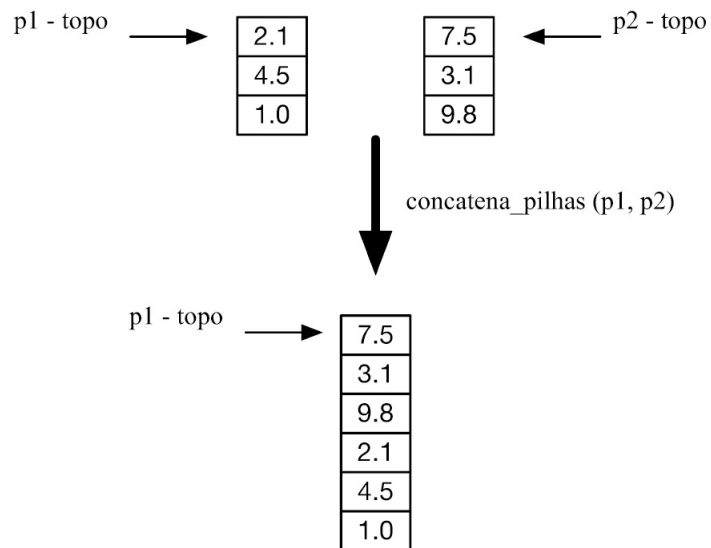
```
typedef struct fila Fila;
Fila* fila_cria(void );
void fila_insere (Fila* f, float v);
float fila_retira (Fila* f);
int fila_vazia (Fila* f);
void fila_libera (Fila* f);
```

Sem conhecer a representação interna desse tipo abstrato e usando apenas as funções declaradas no arquivo de interface, implemente uma função que receba três filas, **f_res**, **f1**, **f2**, e transfira alternadamente os elementos de **f1** e **f2** para **f_res**, conforme ilustrado na figura a seguir:



Note que, ao final dessa função, as filas **f1** e **t2** vão estar vazias e a fila **f_res** vai conter todos os valores que estavam originalmente em **f1** e **f2** (inicialmente **f_res** pode ou não estar vazia). Se uma fila for maior que a outra, os valores excedentes devem ser transferidos para a nova fila no final.

19. Implemente uma função que recebe duas pilhas **p1** e **p2** e passa todos os elementos da pilha **p2** para a 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.



Implemente uma versão usando recursividade e uma versando usando uma terceira pilha.