BCC202 – Estruturas de Dados I (2023-02)

Departamento de Computação - Universidade Federal de Ouro Preto - MG
Professor: **Pedro Silva** (`www.decom.ufop.br/`)

AULA PRÁTICA 07

- Data de entrega: Até 24 de novembro às 23:55.

- Procedimento para a entrega:.

1. Submissão: via ***Moodle***.
2. Os nomes dos arquivos e das funções devem ser especificados considerando boas práticas de programação.
3. Funções auxiliares, complementares aquelas definidas, podem ser especificadas e implementadas, se necessário.
4. A solução deve ser devidamente modularizada e separar a especificação da implementação em arquivos *.h* e *.c* sempre que cabível.
5. Os arquivos a serem entregues, incluindo aquele que contém *main()*, devem ser compactados (*.zip*), sendo o arquivo resultante submetido via ***Moodle***.
6. Caracteres como acento, cedilha e afins não devem ser utilizados para especificar nomes de arquivos ou comentários no código.
7. Siga atentamente quanto ao formato da entrada e saída de seu programa, exemplificados no enunciado.
8. Durante a correção, os programas serão submetidos a vários casos de testes, com características variadas.
9. A avaliação considerará o tempo de execução e o percentual de respostas corretas.
10. Eventualmente, serão realizadas entrevistas sobre os estudos dirigidos para complementar a avaliação.
11. Considere que os dados serão fornecidos pela entrada padrão. Não utilize abertura de arquivos pelo seu programa. Se necessário, utilize o redirecionamento de entrada.
12. Os códigos fonte serão submetidos a uma ferramenta de detecção de plágios em software.
13. Códigos cuja autoria não seja do aluno, com alto nível de similaridade em relação a outros trabalhos, ou que não puder ser explicado, acarretará na perda da nota.
14. Códigos ou funções prontas específicos de algoritmos para solução dos problemas elencados não são aceitos.
15. Não serão considerados algoritmos parcialmente implementados.

- Bom trabalho!

Trabalhando com uma pilha dupla e uma fila

Escreva um algoritmo que leia até 1.000 valores inteiros. O valor 0 (zero) finaliza a entrada de dados. Para cada valor lido, determine se ele é par ou ímpar. Se o número for par, então inclua-o na PILHA PAR (primeira parte da pilha) e, se for ímpar, inclua-o na PILHA ÍMPAR (segunda parte da pilha). **As duas pilhas devem ficar armazenadas no mesmo vetor**. A Figura 1 mostra um exemplo de como manter as duas pilhas.

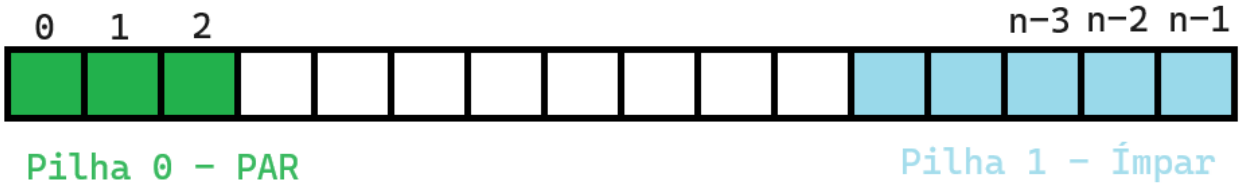

Figura 1: Exemplo de duas pilhas em único vetor.

Após o término da entrada de dados, você deve retirar um elemento de cada pilha alternadamente (iniciando-se pela PILHA PAR) até que ambas as pilhas estejam vazias. Se o elemento retirado de uma das pilhas for um valor

positivo, então você deve incluí-lo em uma FILA, caso contrário, remover um elemento da FILA. Finalmente, imprima o conteúdo da FILA resultante.

Considerações

- Não altere o nome dos arquivos.
- O arquivo `.zip` deve conter na sua raiz somente os arquivos-fonte.
- Há vários casos de teste. Você terá acesso (entrada e saída) de casos específicos para realizar os seus testes localmente.

O código-fonte deve ser modularizado corretamente conforme os arquivos de protótipo fornecidos. As funções de fila são aquelas vistas em sala. As de pilha são semelhantes, mas não iguais. A função `PilhaDuplaCria` aloca uma pilha (que tem um vetor estático de itens dentro) que terá internamente as duas pilhas dentro dela. A função `PilhaDuplaDestroi` libera a pilha criada. As funções `PilhaEhVazia` e `PilhaEhCheia` recebem a pilha dupla e um `id` para identificar qual pilha será avaliada (0 - par e 1 - ímpar) e retornam se a pilha especificada está vazia ou não e cheia ou não respectivamente. As funções `PilhaPush`, `PilhaPop`, `PilhaTamanho` e `PilhaImprime` recebem um `id` para identificar em qual pilha será feita a operção.

Você deve ler os números um em cada linha. A leitura termina quando 0 (zero) for lido. Após o término da leitura, realizar o processamento necessário e imprimir a fila resultante. **A função de imprimir uma fila já está implementada**.

Entrada	**Saída**
1 2 3 4 5 -1 -2 -3 -4 -5 0	`Fila: [(2) (5) (3) (1) ]`

Diretivas de Compilação

```
$ gcc -c pilhadupla.c -Wall
$ gcc -c fila.c -Wall
$ gcc -c pratica.c -Wall
$ gcc pilhadupla.o fila.o pratica.o -o exe
```

Avaliação de *leaks* de memória

Uma forma de avaliar se não há *leaks* de memória é usando a ferramenta valgrind. Um exemplo de uso é:

```
gcc -g -o exe *.c -Wall; valgrind --leak-check=yes -s ./exe < casoteste.in
```

Espera-se uma saída com o fim semelhante a:

```
==38409== ERROR SUMMARY: 0 errors from 0 contexts (suppressed:  0 from 0)
```

Para instalar no Linux, basta usar: `sudo apt install valgrind`.