Estruturas de dados

Trabalho 1

Enunciado

Escreva um programa em C para simular algumas operações de um aeroporto que tem apenas uma pista de decolagem. Os aviões entram sempre no final da pista, e decolam sempre do início da pista.

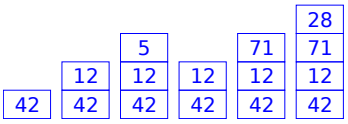
Cada avião tem um bagageiro, que consiste em uma pilha de malas. As malas são sempre colocadas uma em cima da outra.

Seu programa deve ler uma sequencia de comandos do usuário. Cada comando pode ser:

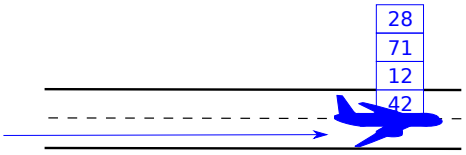
- `load`: cria um novo avião. Após o comando, leia uma sequência de inteiros terminada em 0 (que não faz parte da sequência). Para cada número lido $x \neq -1$, insira uma mala de x kg no bagageiro do avião. Entretanto, se $x = -1$, *retire* uma mala do bagageiro e indique o extravio na saída do programa (na forma `x kg extraviados!`). Como exemplo, considere o seguinte comando:

  ```
  load
  42 12 5 -1 71 28 0
  ```

  A figura abaixo demostra o bagageiro a cada carregamento. Note que uma mala de 5 kg é extraviada:

  Após carregar o avião, ele é inserido no final da pista de decolagem:

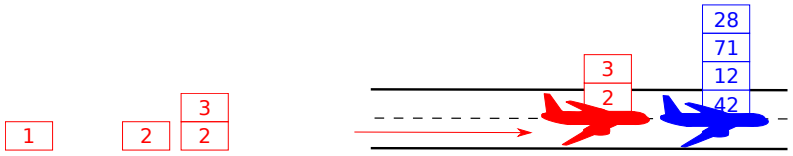
  

  Neste momento, seu programa deve imprimir `ready`.

Ainda como exemplo, considere agora o seguinte comando:

```
load
1 -1 2 3 0
```

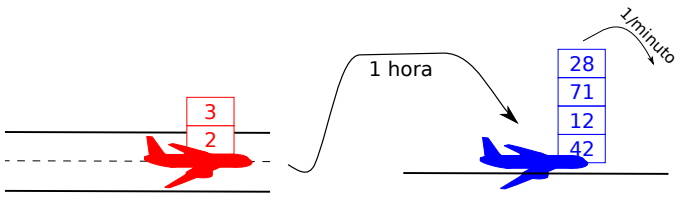
As figuras abaixo demonstram o bagageiro do novo avião (1 kg extraviado), e o mesmo inserido ao final da pista de decolagem:



– `takeoff`: o avião no início da pista decola. Após o comando, leia um inteiro h, indicando que o avião decolou às h horas em ponto. Considere que toda viagem tem exatamente 1 hora de duração. Ao chegar no destino, o bagageiro do avião será esvaziado. Será desempilhada uma mala por minuto. Para cada mala, seu programa deve imprimir o horário em que é desempilhada (no formato hmm) e o peso da mala. Como exemplo, considere o seguinte comando:

```
takeoff
9
```

O comando indica que o avião no início da pista decola às 9 horas. Desta forma, a primeira mala é desempilhada às 10:00; a segunda é desempilhada às 10:01; e assim por diante:



Logo, neste caso, seu programa deve imprimir:

```
1000 28
1001 71
1002 12
1003 42
```

Após o esvaziamento do bagageiro, o avião deixa de existir na simulação.

– `end`: termina a execução do programa.

Exemplo de entrada	**Exemplo de saída**

`load`
`42 12 5 -1 71 28 0`
`load`
`1 -1 2 3 0`
`takeoff`
`9`
`load`
`10 2 -1 0`
`takeoff`
`11`
`takeoff`
`14`
`end`
	`5 kg extraviados!`
	`ready`
	`1 kg extraviados!`
	`ready`
	`1000 28`
	`1001 71`
	`1002 12`
	`1003 42`
	`2 kg extraviados!`
	`ready`
	`1200 3`
	`1201 2`
	`1500 10`

Durante o carregamento do bagageiro, se for lido -1 com o bagageiro vazio, imprima `bagageiro vazio` e continue a simulação normalmente. Ainda, se um comando `takeoff` (e seu horário) é lido com a pista vazia, imprima `pista vazia` e continue a simulação normalmente.

Exemplo de entrada	**Exemplo de saída**

`load`
`42 -1 -1 20 0`
`takeoff`
`8`
`takeoff`
`9`
`end`
	`42 kg extraviados!`
	`bagageiro vazio`
	`ready`
	`900 20`
	`pista vazia`

Ainda, todo bagageiro tem capacidade máxima de 20 itens, assim como a pista pode ter no máximo 10 aviões simultaneamente. Se, durante um comando `load`, for lido um valor $x \neq -1$ com o bagageiro cheio, imprima `bagageiro cheio`, ignore o valor lido, e continue a simulação normalmente. Após o carregamento de um avião, se a pista de decolagem estiver lotada, imprima `pista lotada`, ignore o avião (*liberando toda sua memória usada* e **não** imprimindo `ready`), e continue a simulação normalmente.

Exemplo de entrada	**Exemplo de saída**

`load`
`1 0`
`load`
`2 0`
`load`
`3 0`
`load`
`4 0`
`load`
`5 0`
`load`
`6 0`
`load`
`7 0`
`load`
`8 0`
`load`
`9 0`
`load`
`10 0`
`load`
`11 2 3 4 5 6 7 8 9 10 11 12 13`
`14 15 16 17 18 19 20 21 -1 0`
`end`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`ready`
	`bagageiro cheio`
	`20 kg extraviados!`
	`pista lotada`

Por fim, você pode assumir que nenhum avião decola depois das 22 horas (mas é possível decolar, por exemplo, à 1 hora do dia seguinte).

Implementação

Como principal estrutura de dados, utilize uma *fila de pilhas*. O trabalho deve conter os seguintes arquivos:

- `PilhaEstatica.{h,c}`: definição e implementação de uma pilha estática;
- `PilhaDinamica.{h,c}`: definição e implementação de uma pilha dinâmica;
- `FilaEstatica.{h,c}`: definição e implementação de uma fila estática;
- `FilaDinamica.{h,c}`: definição e implementação de uma fila dinâmica;
- `main.c`: programa principal. Deve incluir (via `#include`) `PilhaEstatica.h` *ou* `PilhaDinamica.h`, *e* `FilaEstatica.h` *ou* `FilaDinamica.h`.

O programa principal deve utilizar fila e pilhas como estrutura *abstrata* de dados. Em particular, deve ser possível "escolher" entre usar uma pilha estática ou uma pilha dinâmica (idem para fila) *apenas alterando os* `#include` *e recompilando de acordo.*

Independentemente da implementação, certifique-se que toda memória alocada por seu programa é desalocada ao final do mesmo (mesmo se ainda houver aviões na pista após a simulação).

Orientações

- O trabalho pode ser feito por equipes de *até* 2 (dois) estudantes;
- Submeta, via *Moodle*, um pacote (zip ou tar.gz) contendo os 9 arquivos citados acima, além de um arquivo de texto (txt) onde conste:
  - O nome de todos os integrantes da equipe;
  - Toda informação que a equipe julgar relevante para a correção (como *bugs* conhecidos, detalhes de implementação, escolhas de projeto, etc.)
- Comente adequadamente seus códigos para facilitar a correção.
- Atenção: a correção será parcialmente automatizada, e a saída do programa será testada com outras entradas além das fornecidas como exemplo. *Siga* ***fielmente*** *o formato de saída dado nos exemplos*, sob pena de grande redução da nota;
- Certifique-se que seu programa funciona antes de submetê-lo;
- O trabalho deve ser entregue até **19 de Maio de 2019 (domingo), 23:59**, via *Moodle*. É suficiente que o trabalho seja submetido por apenas um estudante da equipe;
- Eventuais problemas podem requerer a defesa do trabalho pela equipe.