Lista de Exercícios - Curso de Estruturas de Dados I (em C/C++)
Prof. Igor Machado Coelho - Tópico: Estruturas Lineares

Observação: os exercícios devem ser feitos em C/C++ (ou similar!). Foque mais na lógica do que em erros básicos de programação e SEMPRE discuta a proposta do algoritmo (não quero apenas código!). Sempre analise a complexidade assintótica dos métodos implementados.

1. Considere um tipo chamado Deque, que inclui manipulação de dois extremos em uma estrutura linear (como se operasse como Pilha e Fila simultaneamente).

```
template<typename Agregado, typename Tipo>
concept bool DequeTAD = requires(Agregado a, Tipo t){
  // requer operação de consulta ao elemento 'inicio'
 { a.inicio() };
  // requer operação de consulta ao elemento 'fim'
 { a.fim() };
  // requer operação 'insereInicio' sobre tipo 't'
 { a.insereInicio(t) };
  // requer operação 'insereFim' sobre tipo 't'
 { a.insereFim(t) };
  // requer operação 'removeInicio' e retorna tipo 't'
 { a.removeInicio() };
  // requer operação 'removeFim' e retorna tipo 't'
 { a.removeFim() };
};
```

1.a) Satisfaça as seguintes operações de um DequeTAD para o tipo ‘char’, utilizando uma estrutura Sequencial OU uma estrutura encadeada:

```
class Deque {
   public:
   // implementar métodos propostos no TAD Deque
};
static_assert(DequeTAD<Deque, char>); // testa se Deque está correto
```

1.b*) Implemente uma estrutura PilhaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

```
class PilhaDeque{
   public:
   Deque d; // Deque para ‘char’ (veja exercício anterior)
   // SOMENTE espaço auxiliar CONSTANTE aqui (nenhum vetor, lista, etc)
   // implementar métodos do TAD Pilha
};
static_assert(DequeTAD<Deque, char>); // testa se Deque está correto
static_assert(PilhaTAD<PilhaDeque, char>); // testa se Pilha está correta
```

1.c*) Implemente uma estrutura FilaDeque para tipo ‘char’, utilizando somente um Deque como armazenamento interno e mais espaço auxiliar constante:

```
class FilaDeque{
   public:
   Deque d; // Deque para ‘char’ (veja exercício anterior)
   // SOMENTE espaço auxiliar CONSTANTE aqui (nenhum vetor, lista, etc)
   // implementar métodos do TAD Fila
};
static_assert(DequeTAD<Deque, char>); // testa se Deque está correto
static_assert(FilaTAD<FilaDeque, char>); // testa se Fila está correta
```

2) Implemente uma estrutura que satisfaz o TAD Pilha para o tipo ‘char’ e somente utiliza duas Filas como armazenamento interno (mais espaço constante):

```
#include<queue> // Fila genérica em C++
class Pilha2F{
   public:
   std::queue<char> f1; // Fila para ‘char’
   std::queue<char> f2; // Fila para ‘char’

   // SOMENTE espaço auxiliar CONSTANTE aqui (nenhum vetor, lista, etc)
   // implementar métodos do TAD Pilha
};
static_assert(PilhaTAD<Pilha2F, char>); // testa se Pilha está correta
```

3) Implemente uma estrutura que satisfaz o TAD Fila para o tipo ‘char’ e somente utiliza duas Pilhas como armazenamento interno (mais espaço constante):

```
#include<stack> // Pilha genérica em C++
class Fila2P{
   public:
   std::stack<char> p1; // Pilha para ‘char’
   std::stack<char> p2; // Pilha para ‘char’

   // SOMENTE espaço auxiliar CONSTANTE aqui (nenhum vetor, lista, etc)
   // implementar métodos do TAD Fila
};
static_assert(FilaTAD<Fila2P, char>); // testa se Fila está correta
```

4) Escreva um algoritmo que dada uma pilha padrão P externa passada como parâmetro, inverte o conteúdo de P. Somente utilize as estruturas extras permitidas como armazenamento externo(mais espaço constante)

a) Uma Fila

```
void inverte(std::stack<char>* p) {
   std::queue<char> f; // somente essa fila e mais espaço auxiliar constante
}
```

b) Duas Pilhas

```
void inverte(std::stack<char>* p) {
   std::stack<char> p1; // primeira pilha auxiliar
   std::stack<char> p2; // segunda pilha auxiliar
   // mais espaço auxiliar constante
}
```

c) Uma Pilha

```
void inverte(std::stack<char>* p) {
   std::stack<char> p1; // uma pilha auxiliar
   // mais espaço auxiliar constante
}
```

5) Escreva um algoritmo que dada uma fila padrão F externa passada como parâmetro, inverte o conteúdo de F. Somente utilize as estruturas extras permitidas como armazenamento externo (mais espaço constante)

a) Uma Pilha

```
void inverte(std::queue<char>* f) {
   std::stack<char> p; // somente essa pilha e mais espaço auxiliar constante
}
```

b) Duas Filas

```
void inverte(std::queue<char>* f) {
   std::queue<char> f1; // primeira fila auxiliar
   std::queue<char> f2; // segunda fila auxiliar
   // mais espaço auxiliar constante
}
```

6) Criar uma implementação do TAD Pilha para o tipo 'int', chamada PilhaMin, que oferece os métodos do TAD e também o método obterMinimo(), que retorna o menor elemento da pilha. O método obterMinimo() deve operar em tempo constante.

```
class PilhaMin {
  public:
  // incluir variáveis necessárias
  int topo();
  int desempilha();
  void empilha(int t);
  int obterMinimo();
  // mais métodos auxiliares
}
static_assert(PilhaTAD<PilhaMin, int>); // testa de PilhaMin está de acordo com o TAD
```

7) Escreva um algoritmo que converte uma expressão aritmética parentizada usando as 4 operações para a expressão correspondente em notação polonesa reversa.
Exemplo:
Entrada: "((A+B)*(C-(F/D)))"
Saída: "AB+CFD/-*"

```
// 'expressao' eh um string terminado em '\0', com tamanho N (sem contar o \0)
// 'saida_polonesa' eh um string com capacidade máxima N
void polonesa(char* expressao, int N, char* saida_polonesa) {
   // escreva o resultado no string 'saida_polonesa'
}
```