Lista 01

1. O que é um Tipo Abstrato de Dados (TAD)? Quais são as características fundamentais?

2. Quais as vantagens de se programar utilizando o conceito de TADs? Explique com exemplos.

3. Um estacionamento com N (N = 200) vagas deseja manter a relação dos carros que estão estacionados (ao entrar no estacionamento o carro é cadastrado no sistema, e ao sair é removido). Criar um TAD para gerenciar o estacionamento. O TAD deve ter operações para inserção (na entrada do carro), remoção (na saída do carro), alteração (caso algum cadastro tenha sido feita de forma incorreta) e busca (para verificar se um carro está estacionado). O cadastro deve conter os seguintes dados:

   - Placa: placa do carro. Será usado como chave de busca. Não podem existir dois carros com a mesma placa (string 8);
   - Marca/Modelo: marca e modelo do carro (string 30). Ex.: VW/Gol GL;
   - Cor: cor do carro (string 20);

4. Por que a seguinte versão de insere não funciona?

```
void insere (int x, No *p) {
  No novo;
  novo.info = x;
  novo.prox = p->prox;
  p->prox = &novo;
}
```

5. Escreva uma função recursiva e outra não recursiva para contar o número de elementos na lista ligada apontada por `p`:

```
int nElementos(No* p);
```

6. Escreva uma função recursiva e outra não recursiva para encontrar o nó com conteúdo mínimo na lista ligada apontada por `p`:

```
No* menorElemento(No* p);
```

7. Escreva uma função recursiva e outra não recursiva que inverte a lista ligada apontada por `p`. Nenhum nó auxiliar deve ser criado.

```
No* inverteLista(No* p);
```

8. Considere que os elementos da lista ligada estão ordenados. Escreva uma função que remove um elemento com valor v da lista l. Caso nenhum elemento com este valor seja encontrado, a função deve retornar 0.

```
int remove(No** l, int v);
```

9. No exercício anterior, por que é necessário utilizar um parâmetro do tipo `No**`? Se a lista tivesse um nó cabeça, um parâmetro do tipo `No*` seria adequado?

10. Considerando a declaração fornecida (com `No**`) você consegue escrever uma versão recursiva desta rotina?

11. Escreva uma função que recebe duas listas `x`= $(x_1, \ldots, x_n)$ e `y`= $(y_1, \ldots, y_m)$ como parâmetro e retorna uma lista formada pelos elementos de x e y intercalados. A lista resultante será da forma $(x_1, y_1, \ldots, x_m, y_m, x_{m+1}, \ldots, x_n)$ se $m \leq n$ ou $(x_1, y_1, \ldots, x_n, y_n, y_{n+1}, \ldots, y_m)$ se $m > n$. Nenhum nó adicional deve ser criado e as listas `x` e `y` devem ficar vazias ao final (receber NULL).

```
No* intercalaListas(No** x, No** y);
```

    Escreva uma versão recursiva e outra não recursiva.

12. Escreva uma função que receba uma lista duplamente encadeada e devolva o endereço de um nó que esteja o mais próximo possível do meio da lista. Faça isso sem contar explicitamente o número de nós da lista.

13. Implemente uma função que converte um número decimal para binário. Utilize pilha.

14. Escreva uma função que tranforma uma expressão da notação infixa para a notação posfixa.

15. Converta as expressões na notação infixa para as formas pós-fixa e pré-fixa:

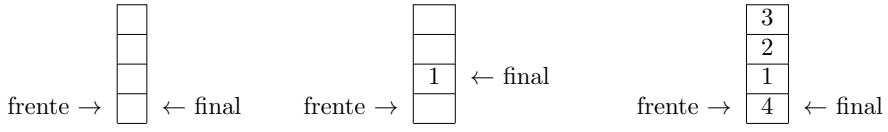
    (a) $D - B + C$
    (b) $A * B + C * D$
    (c) $(A + B) * C - D * F + C$
    (d) $(A - 2 * (B + C) - D * E) * F$

16. Converta as expressões da notação pós-fixa ou pré-fixa em infixa:

    (a) $AB * C - D+$
    (b) $ABC + *D-$
    (c) $+ - *ABCD$
    (d) $- * A + BCD$

17. Escreva uma função que calcula o valor de uma expressão pós-fixa. Utilize os valores $A = 2$, $B = 3$, $C = 4$ e $D = 5$ para verificar o resultado das expressões do último exercício.

18. Considere uma implementação de fila utilizando vetor circular, que utiliza apontadores (índices) para a frente e o final da fila: `frente` aponta para a posição imediatamente anterior ao primeiro elemento da fila e `final` aponta para o último elemento inserido, se existir. Comente a dificuldade para se diferenciar fila cheia de fila vazia.



(a) Fila vazia (b) Fila com um elemento (c) Fila com quatro elementos

    Implemente as funções `cria_fila`, `fila_vazia`, `insere_file` e `remove_fila` utilizando um *flag* (variável booleana) para indicar se a fila está cheia ou vazia.

19. Uma fila *simétrica*, ou do inglês *doubled-ended queue (deque)*, permite inserção e remoção em ambas as extremidades. Faça as declarações de tipo apropriadas e escreva as seguintes rotinas para implementação utilizando lista ligada de filas simétricas: `cria_fila`, `fila_vazia`, `insere_frente`, `insere_final`, `remove_frente` e `remove_final`.

    Considere a seguinte restrição adicional: todas estas operações devem ser implementadas em tempo constante, ou seja, nenhuma precisa percorrer todos os elementos da fila para ser executada. Você consegue escrever uma implementação que respeita esta restrição utilizando listas ligadas simples? E utilizando listas duplamente ligadas?

20. Podemos aproveitar uma implementação para fila simétrica para implementar uma fila ou uma pilha. Mostre como isso poderia ser feito.