Centro de Matemática, Computação e Cognição – UFABC
MCTA028-15 – Programação Estruturada – 3º Quadrimestre de 2023
http://professor.ufabc.edu.br/~m.sambinelli/courses/2023Q3-PE/index.html

Laboratório 6: lista ligada

Atenção: de agora em diante está terminantemente **proibido** usar arrays estáticos de tamanho variado como os exibidos abaixo:

```
int n;
int vetor[n]; // nunca definir a dimensão de um array com uma variável
```

Essa proibição aplica-se a tudo: listas, trabalho e avaliação. Qualquer programa usando o tipo de construção acima receberá nota **zero**. Todo array dinâmico deverá ser construído usando alocação dinâmica de memória:

```
int* vetor = calloc(n, sizeof(int)); // ou
int* vetor2 = malloc(n * sizeof(int));
```

Instruções

- Em todos os seus programas você deve gerenciar corretamente a memória, liberando toda a memória requerida pelo seu programa após o término do uso. Programas com vazamento de memória receberão uma penalização de 25% do valor da nota total do exercício. Você pode verificar se o seu programa possui vazamento de memória com o comando

  ```
  valgrind --leak-check=full /caminho/para/o/seu/programa
  ```

  Para que o comando acima funcione, você deve habilitar a *flag* de *debug* do seu compilador. O exemplo a seguir ilustra como deve ser feito caso você use o *gcc* para compilar o seu programa

  ```
  gcc -Wall -Wextra -Wvla -g -std=c99 arquivo.c
  ```

- Em vários exercícios, eu peço a vocês para que escrevam uma função de um determinado tipo. Além de escrever essa função, vocês também devem escrever uma função `main()` que irá usar essa função com dados fornecidos pelo usuário. Ou seja, a sua `main()` deverá pedir a entrada para o usuário e passar esses dados como parâmetro para a função que você desenvolveu. Requisite esses dados imprimindo mensagens na tela, para que o professor saiba o que digitar quando estiver corrigindo o seu trabalho.

Questão 1. Em aula, vimos uma implementação recursiva da função `void destruir_lista(Lista l)`. Agora, apresente uma versão iterativa dessa função.

Questão 2. Faça uma função que busca um elemento `x` em uma lista ligada, devolvendo o ponteiro para o nó encontrado ou `NULL` se o elemento não existir na lista.

Questão 3. Faça uma função iterativa que remove a primeira ocorrência (se existir) de um elemento `x` de uma lista ligada dada.

Questão 4. Faça uma função recursiva que remove a primeira ocorrência (se existir) de um elemento `x` de uma lista ligada dada.

Questão 5. Faça uma função que remove todas as ocorrências de um elemento `x` de uma lista ligada dada.

Questão 6. Usando a estrutura `Lista` definida em sala de aula, crie uma função `void insert(Lista *l, int x)` que insere um elemento `x` em uma lista `l`. Observe que, diferentemente dos exemplos vistos em sala de aula, você não pode retornar o endereço do início da nova lista. É por isso que, ao invés de receber `Lista l`, você está recebendo `Lista *l`!

Questão 7. Faça uma função que receba duas listas, digamos `l1` e `l2`, como parâmetro e concatene a lista `l2` no final da lista `l1`.

Questão 8. Implemente a função `Lista inverte(Lista l)` que retorna um lista `l` com os elementos invertidos. Mais precisamente, se os elementos da lista `l` estão na ordem `a, b, c, d, e`, então os elementos da lista retornada devem estar na ordem `e, d, c, b, a`.