CI1001 - Programação I

André Grégio, Fabiano Silva, Luiz Albini e Marcos Castilho
Departamento de Informática – UFPR, Curitiba/PR

Oitava lista de exercícios

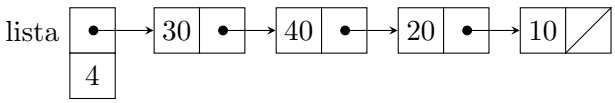
Tipo Abstrato de Dados Lista
Implementação de uma possível biblioteca

Escreva uma biblioteca em `C` de nome `lista.c`, que inclua um arquivo de *headers* `lista.h` (que será fornecido), que implemente funções que manipulem uma lista ligada de nodos, da seguinte maneira:

Você deve definir dois tipos:

- O tipo *t_nodo* é uma estrutura que contém dois campos, o primeiro é um elemento do tipo *int*, o segundo é um apontador para o próprio tipo *t_nodo*;
- O tipo *t_lista* é uma estrutura que contém dois campos, o primeiro é um apontador para o início da lista de nodos do tipo *t_nodo* e o segundo é um inteiro que contém número de elementos da lista (seu tamanho), no caso do exemplo é 4, pois a lista contém os números 30, 40, 20 e 10;

A ilustração abaixo mostra um esquema do relacionamento destas estruturas (a caixa contendo uma barra na diagonal representa `NULL`):



Uma vez definidos estes tipos, implemente as funções cujos protótipos são os seguintes:

- `int cria_lista(t_lista *l);`
  Cria uma lista vazia, isto é, aponta para `NULL` e contém tamanho zero; Retorna 1 se a operação foi bem sucedida e zero caso contrário;
- `int lista_vazia(t_lista *l);`
  Retorna 1 se a lista está vazia e zero caso contrário;
- `void destroi_lista(t_lista *l);`
  Remove todos os elementos da lista e faz com que ela aponte para `NULL`;
- `int insere_inicio_lista(int x, t_lista *l);`
  Insere o elemento x no início da lista; Retorna 1 se a operação foi bem sucedida e zero caso contrário;
- `int insere_fim_lista(int x, t_lista *l);`
  Insere o elemento x no final da lista; Retorna 1 se a operação foi bem sucedida e zero caso contrário;
- `int insere_ordenado_lista(int x, t_lista *l);`
  Insere o elemento x na lista de maneira que ela fique em ordem crescente, do início para o final dela; Retorna 1 se a operação foi bem sucedida e zero caso contrário;
- `void imprime_lista(t_lista *l);`
  Imprime os elementos da lista, do início ao final. Se ela for vazia não imprime nada, nem um `\n`;

- `int remove_primeiro_lista(int *item, t_lista *l);`
  Remove o primeiro elemento da lista e o retorna em `*item`; Retorna 1 se a operação foi bem sucedida e zero caso contrário;

- `int remove_ultimo_lista(int *item, t_lista *l);`
  Remove o último elemento da lista e o retorna em `*item`; Retorna 1 se a operação foi bem sucedida e zero caso contrário;

- `int remove_item_lista(int chave, int *item, t_lista *l);`
  Se o elemento `chave` existir na lista, o retorna em `*item`; Retorna 1 se a operação foi bem sucedida e zero caso contrário (elemento não encontrado também retorna zero);

- `int pertence_lista(int chave, t_lista *l);`
  Retorna 1 se o elemento contendo a chave `chave` existe na lista, caso contrário retorna zero;

- `int concatena_listas(t_lista *l, t_lista *m);`
  Concatena os elementos da lista `m` (na mesma ordem) ao final da lista `l`. Faz a lista `m` conter o valor `NULL`;

- `int copia_lista(t_lista *l, t_lista *m);`
  Cria uma nova lista `m` contendo uma cópia exata da lista `l`. Retorna 1 se a operação foi bem sucedida e zero caso contrário.

Seu programa deve ser testado usando-se o arquivo `testa.c` que é fornecido juntamente com este enunciado.

- Compile com as opções `-ansi -Wall` do *gcc*;

- Sua implementação deve garantir que todo espaço de memória alocado dinamicamente deve ser liberado quando não for mais necessário:

  - Use o programa `valgrind` para garantir que este item seja atendido!!! Um exemplo de saída para correta para o `valgring` está no final deste texto;

- Faça uma boa apresentação do código (legibilidade, identação);

- Use bons nomes para as variáveis;

- Faça uso de constantes que facilitem a manutenção do código, bem como a legibilidade;

- Use corretamente as variáveis locais e globais;

- Bons comentários no código são fundamentais, mas não exagere no número deles, use com bom senso em situações pertinentes.

```
ci1001@fradim:~/tmp/$ valgrind ./testa
==5051== HEAP SUMMARY:
==5051==     in use at exit: 0 bytes in 0 blocks
==5051==   total heap usage: 23 allocs, 23 frees, 1,376 bytes allocated
==5051==
==5051== All heap blocks were freed -- no leaks are possible
==5051==
==5051== For counts of detected and suppressed errors, rerun with: -v
==5051== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```