Estrutura de Dados I – 2014.1

prof. Andres Jessé Porfirio

1. Conceito

Uma lista linear é uma sequência de n elementos de um tipo de dado: x_1, x_2, x_3, ..., x_m. A propriedade estrutural envolve as posições relativas dos elementos. Se n >= 0, então:

– x_0 é o primeiro elemento;
– $x_{(n-1)}$ é o último elemento;
– x_i é sucessor de x_{i-1};
– x_i é antecessor de x_{i+1}.

Conforme implementação, listas lineares são muito úteis em aplicações onde número de elementos não é conhecido, a priori.

Exemplos de uso:
– Lista de arquivos de uma pasta;
– Lista de programas instalados;
– Lista de impressoras compartilhadas em uma rede;
– Lista de operações que realizadas;
– Entre outros.

São operações comuns em listas:
– Criar uma lista inicialmente vazia;
– Inserir um elemento na i-ésima posição, desde que: 0 <= i <= n;
– Remover o i-ésimo elemento, 0 <= i <= (n-1);
– Acessar o i-ésimo elemento, 0 <= i <= (n-1);
– Alterar o i-ésimo elemento, 0 <= i <= (n-1);
– Retornar o número de elementos;
– Localizar determinado elemento;
– Excluir todos os elementos da lista.

Operações específicas podem variar conforme o contexto da aplicação. São exemplos:
– Fundir duas listas em uma;
– Dividir uma lista em duas;
– Ordenar os elementos segundo algum critério;
– Verificar se uma lista está ordenada;
– Inverter a ordem dos elementos;
– Trocar a ordem de dois elementos.

2. Implementação Encadeada (Alocação Dinâmica)

Implemente uma lista utilizando alocação dinâmica, este tipo de lista também é conhecido como implementação encadeada (devido ao encadeamento de estruturas, conceito que abordamos algumas aulas atrás). Siga o modelo das implementações anteriores, ou seja, crie uma struct, defina um novo tipo de dado e utilize-o nas funções. Faça com que a lista trabalhe com um tipo de dado “Data”, definido no “.h”. Implemente o exercício no padrão “.h”+”.c” (dynamic_list.h + dynamic_lsit.c).

DynamicList* list_inicialize() //inicializa

Aloca, inicializa e retorna uma nova lista.

int list_is_empty(DynamicList* l) //estaVazia

Retorna 1 se a lista está vazia, 0 caso contrário.

int list_add(DynamicList* l, int index, Data data) //adiciona

Adiciona o inteiro na posição especificada da lista. Perceba que nenhuma outra operação de adição precisa ser implementada. Basta ativar esta, mais genérica, com parâmetros. Retorna 1 em caso de sucesso e 0 caso contrário.

int list_add_first (DynamicList* l, Data data) //adicionaInicio

Adiciona o inteiro no início da lista. Retorna 1 em caso de sucesso, 0 caso contrário.

int list_add_last (DynamicList* l, Data data) //adicionaFinal

Adiciona o inteiro ao final da lista. Retorna 1 em caso de sucesso, 0 caso contrário.

int list_contains(DynamicList* l, Data data) //contem

Retorna 1 se a lista contém o elemento especificado, 0 caso contrário.

Data list_get(DynamicList* l, int index) //retorna

Retorna, mas não remove, o elemento da posição especificada na lista. Considere índices que o usuário pode fornecer um índice inválido ou ainda a lista pode estar vazia, nestes casos retorne um codigo de erro: NULL para ponteiros, -1 para tipos primitivos (defina "ErrorCode" no ".h"). Considere isso para as demais operações. Perceba esta como a operação mais genérica para retornar elementos.

Data list_get_first(DynamicList* l) //retornaPrimeiro

Retorna, mas não remove, o primeiro elemento da lista. Considere que a lista pode estar vazia, neste caso retorne o código de erro.

Data list_get_last(DynamicList* l) //retornaUltimo

Retorna, mas não remove, o último elemento da lista. Considere que a lista pode estar vazia, neste caso retorne o código de erro.

int list_index_of(DynamicList* l, Data data) //primeiraOcorrencia

Retorna o índice da primeira ocorrência do elemento especificado na lista, ou (-1) se a lista não contiver o elemento. Note que você vai precisar comparar elementos, em caso de tipos primitivos pode ser usado o operador "==", para tipos definidos pelo usuário (ex: Data = Student*) deve ser definido um comparador (ex: is_data_equal(Data d1, Data d2), onde d1 e d2 serão estudantes, comparados de acordo com algum critério definido dentro de is_data_equal).

int list_last_index_of(DynamicList* l, Data data) //ultimaOcorrencia

Retorna o índice da última ocorrência do elemento especificado na lista, ou (-1) se a lista não contiver o elemento. Comparador: idem função anterior.

Data list_remove(DynamicList* l, int index) //remove

Remove e retorna o primeiro elemento da posição especificada na lista. Operação mais genérica de remoção. Retorne o código de erro caso o usuário forneça um índice inválido.

Data list_remove_fist(DynamicList* l) //removePrimeiro

Remove e retorna o primeiro elemento da lista. Retorne o código de erro caso o usuário forneça um índice inválido.

Data list_remove_last(DynamicList* l) //removeUltimo

Remove e retorna o último elemento da lista. Retorne o código de erro caso o usuário forneça um índice inválido.

void list_clear(DynamicList* l) //esvazia ou reinicializa

Remove todos os elementos da lista.

int list_size(DynamicList* l) //tamanho

Retorna o número de elementos na lista. Note que esta função pode ser facilitada com o uso de uma variável inteira "size" na struct principal.

void list_print(DynamicList* l) //imprime ou textualiza

Imprime na saída a representação textual da lista. Ex (considerando uma lista de inteiros):

– lista vazia:
{}
– lista com um elemento:
{10}
– lista com mais elementos:
{10, 20, 30, 40}

Note que a impressão de elementos pode variar de acordo com o tipo de dados que a lista armazena, para facilitar a implementação, crie uma função "print_item(Data d)", responsável por imprimir um único elemento. Esta função pode ser utilizada em list_print, assim, independentemente do tipo de dado armazenado na lista, a função print_item será sempre responsável pela impressão dos itens.

3. Referências

Material baseado nas aulas do prof. Eleandro Maschio.