Laboratório 6

Marcos Tidball - 00302962

April 20, 2021

1 Questão 1

A função recebe como entrada uma lista encadeada e imprime os componentes dela. Podemos notar isso devido ao `while PtLista <> nil do`, que itera pela lista ao até que o ponteiro para o próximo item seja `NULL`, que marca o final da lista. Para cada iteração do loop a função imprime informações da `struct` que está em cada nodo da lista. Mais especificamente, a função imprime o `código` e `nome` de cada entrada.

2 Questão 2

Em ordem de acordo com os pontos de interrogação da imagem no pdf do Laboratório 6:

1. `Pt`
2. `NULL`
3. `PtLista`

3 Questão 3

3.1 a

```c
// definindo o struct produto
typedef struct produto {
    int codigo;
    char nome[30];
    float preco;
} produto;

// define como um elemento da lista encadeada eh
typedef struct nodo {
    produto prod; // campo de dado
    struct nodo* prox; // campo do ponteiro que aponta para o proximo elemento
} Nodo;

// define a lista encadeada
typedef struct lista {
    Nodo* inicio; // um ponteiro para o primeiro elemento
} Lista;
```

3.2 b

```c
// inicializa a lista
void inicializarLista (Lista* l) {
    // alocamos memoria para o primeiro nodo da lista
    l->inicio = (Nodo*) malloc(sizeof(Nodo));

    if (l->inicio != NULL) { // caso alocacao deu certo
        l->inicio = NULL; // inicio aponta pra NULL, pois temos uma lista vazia
    }
}

// printa info de cada elemento da lista
void printLista(Lista* l) {
    Nodo* atual = l->inicio;

    while (atual != NULL) {
        printf("%d, %s, %f\n",
        atual->prod.codigo, atual->prod.nome, atual->prod.preco);
        atual = atual->prox;
    }
}
```

4 Questão 4

```c
#include <string.h>

// insere elemento de um modo ordenado na lista
int inserirOrdenado (Lista* l, produto novo) {
    Nodo* nodo_novo = (Nodo*) malloc(sizeof(Nodo));
    if (nodo_novo == NULL) return -1;

    nodo_novo->prod = novo;

    if (checarListaVazia(l)) { // caso lista vazia, inserir no inicio
        nodo_novo->prox = l->inicio; // com lista vazia, isso eh == NULL
        l->inicio = nodo_novo;
    }

    else {
        // cria nodos auxiliares
        Nodo* nodo_anterior;
        Nodo* nodo_atual = l->inicio;

        while (nodo_atual != NULL && strcmp(nodo_atual->prod.nome, novo.nome) < 0) {
            nodo_anterior = nodo_atual;
            nodo_atual = nodo_atual->prox;
        }

        nodo_novo->prox = nodo_atual;
        nodo_anterior->prox = nodo_novo;
    }
```

```
    return 1;
}

// remove elemento com nome especifico
int removerPosicao (Lista *l, char nome_remover[]) {
    if (checarListaVazia(l)) return -1;

    Nodo* nodo_anterior;
    Nodo* nodo_atual = l->inicio;

    while (nodo_atual != NULL && strcmp(nodo_atual->prod.nome, nome_remover) == 0 ) {
        nodo_anterior = nodo_atual;
        nodo_atual = nodo_atual->prox;
    }

    if (nodo_atual == l->inicio) {
        l->inicio = nodo_atual->prox;
    }
    else {
        nodo_anterior->prox = NULL;
        free(nodo_atual);
    }

    return 1;
}

// destroi lista
void liberaLista (Lista* l) {
    if (l->inicio != NULL) { // se lista nao esta vazia
        Nodo* nodo_atual = l->inicio;
        Nodo* nodo_prox;

        while(nodo_atual != NULL) {
            nodo_prox = nodo_atual->prox;
            free(nodo_atual);
            nodo_atual = nodo_prox;
        }
    }

    l->inicio = NULL; // libera o ponteiro para o primeiro elemento
}
```

5 Questão 5

As respostas estão em orderm de acordo com o pdf do Laboratório 6:

1. $O(1)$.
2. $O(1)$.
3. $O(1)$.
4. $O(1)$.

5. $O(1)$.

6. $O(1)$.

Notamos que a pilha, a fila e o deque são estruturas muito eficientes ao se tratar de inserção e remoção (que sempre ocorrerá nas extremidades).