QXD0010 - ESTRUTURA DE DADOS - 01A - 2025.2

Descrição | Enviar | Editar | Visualizar envios

[linked list] Implementar uma lista simplesmente encadeada - Parte 4

Data de entrega: domingo, 7 dez 2025, 23:59
Arquivos requeridos: main.c, LinkedList.h, LinkedList.c (Baixar)
Número máximo de arquivos: 5
Tipo de trabalho: Trabalho individual

Motivação (Listas Simplesmente Encadeadas)

Neste exercício, você deve completar a implementação da lista simplesmente encadeada com novas funções.

Os protótipos das funções que devem ser implementadas estão listados abaixo juntamente com um comentário explicando o que cada função deve fazer.

```
// Esta função recebe como entrada uma lista simplesmente encadeada e
// executa uma varredura na lista, comparando nós consecutivos dois-a-dois e movendo o maior
// elemento que ela encontra para a direita na lista.
// A função retorna como resultado a lista modificada.
// Por exemplo, se passarmos como entrada a lista [9,8,7,6,5] então a lista resultante deve ser
[8,7,6,5,9].
// Veja que a função foi comparando os elementos dois-a-dois da esquerda para a direita, movendo o
maior para a
// direita, nesse caso o número 9 que é o maior de todos na varredura acabou sendo movido para o
final da lista.
// Mais um exemplo: se passarmos a lista [6,5,4,3,8,9] como entrada, então a lista resultante deve
ser [5,4,3,6,8,9].
Node* list_varredura(Node *p);

// Esta função usa a função list_varredura descrita acima para implementar
// o algoritmo de ordenação por bolha (bubblesort) numa lista encadeada.
// Complexidade: O(n^2).
// Não é preciso alocar memória nessa função.
```

```
Node* list_bubblesort(Node *p);
```

Observação 1: As funções possuem restrições que devem ser obedecidas. Se as restrições não forem satisfeitas, haverá redução da nota.

Observação 2: O arquivo **main.c** e o arquivo **LinkedList.h** já foram codificados e foram incluídos nesta atividade.

Resta implementar apenas as funções listadas acima.

O programa principal em main.c lê comandos passados como entrada e manipula uma ou mais listas.

Os comandos aceitos pelo menu principal estão listados abaixo.

Comando	Significado
exit	sair do programa
create	cria uma nova lista vazia
show	mostra todas as listas na tela
bubblesort p	ordena a lista p usando o algoritmo de ordenação por bolhas
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio da lista k

Arquivos requeridos

main.c

```c
// ESTE ARQUIVO JA ESTA PRONTO, NAO MEXA NELE
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include "LinkedList.h"

// Vetor dinâmico simples
typedef struct {
    Node **data;
    size_t size;
} Vector;

// Inicializa vetor
void vector_init(Vector *v) {
    v->data = NULL;
    v->size = 0;
}

// Adiciona uma lista nova no final
void vector_push_back(Vector *v, Node *list) {
    v->data = realloc(v->data, (v->size + 1) * sizeof(Node*));
    v->data[v->size] = list;
    v->size++;
}

// Libera todas as listas
void vector_clear(Vector *v) {
    for(size_t i = 0; i < v->size; i++) {
        list_free(v->data[i]);
    }
    free(v->data);
    v->data = NULL;
    v->size = 0;
}

int main() {
    Vector listas;
    vector_init(&listas);

    char comando[300];

    while (true) {

        if(!fgets(comando, sizeof(comando), stdin))
            break;

        // Remove \n
        comando[strcspn(comando, "\n")] = 0;

        printf("$%s\n", comando);

        char *token = strtok(comando, " ");

        if(token == NULL)
            continue;

        // -------- exit --------
        if(strcmp(token, "exit") == 0) {
            vector_clear(&listas);
            break;
        }

        // -------- create --------
        else if(strcmp(token, "create") == 0) {
            vector_push_back(&listas, list_create());
        }

        // -------- size l --------
        else if(strcmp(token, "size") == 0) {
            int l = atoi(strtok(NULL, " "));
            printf("size list %d: %zu\n", l, list_size(listas.data[l]));
        }

        // -------- varredura l --------
        else if(strcmp(token, "varredura") == 0) {
            int l = atoi(strtok(NULL, " "));
            listas.data[l] = list_varredura(listas.data[l]);
        }

        // -------- bubblesort l --------
        else if(strcmp(token, "bubblesort") == 0) {
            int l = atoi(strtok(NULL, " "));
            listas.data[l] = list_bubblesort(listas.data[l]);
        }

        // -------- addFront l a1 a2 ... --------
        else if(strcmp(token, "addFront") == 0) {
            int l = atoi(strtok(NULL, " "));
            char *p = NULL;
            while((p = strtok(NULL, " ")) != NULL) {
                int val = atoi(p);
                listas.data[l] = list_push_front(listas.data[l], val);
            }
        }

        // -------- addBack l a1 a2 ... --------
        else if(strcmp(token, "addBack") == 0) {
            int l = atoi(strtok(NULL, " "));
            char *p = NULL;
            while((p = strtok(NULL, " ")) != NULL) {
                int val = atoi(p);
                listas.data[l] = list_push_back(listas.data[l], val);
            }
        }

        // -------- show --------
        else if(strcmp(token, "show") == 0) {
            for(size_t i = 0; i < listas.size; i++) {
                printf("lista %zu: ", i);
                list_print(listas.data[i]);
            }
        }

        else {
            printf("comando inexistente\n");
```

```
        }
    }

    return 0;
}
```

LinkedList.h

```c
// ESTE ARQUIVO JA ESTA PRONTO, NAO MEXA NELE
#ifndef LINKED_LIST_H
#define LINKED_LIST_H
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct node Node;

// Função que cria uma lista vazia
// sem nó sentinela
Node *list_create(void);

// Insere um valor no início da lista
// e retorna a lista modificada.
Node *list_push_front(Node *list, int value);

// Insere um valor ao final da lista
// e retorna a lista modificada.
Node *list_push_back(Node *list, int value);

// Função que recebe um ponteiro para a cabeça da lista e
// imprime o conteúdo dos elementos na tela.
// Formato: [ a1 a2 a3 ... an ]
void list_print(Node *list);

// Função que libera todos os nós
// alocados dinamicamente
void list_free(Node *list);

// Função que imprime o tamanho da lista (número de nós)
size_t list_size(Node *list);

// Esta função recebe como entrada uma lista simplesmente encadeada e
// executa uma varredura na lista, comparando nós consecutivos dois-a-dois e movendo o maior
// elemento que ela encontra para a direita na lista.
// A função retorna como resultado a lista modificada.
// Por exemplo, se passarmos como entrada a lista [9,8,7,6,5] então a lista resultante deve ser [8,7,6,5,9].
// Veja que a função foi comparando os elementos dois-a-dois da esquerda para a direita, movendo o maior para a
// direita, nesse caso o número 9 que é o maior de todos na varredura acabou sendo movido para o final da lista.
// Mais um exemplo: se passarmos a lista [6,5,4,3,8,9] como entrada, então a lista resultante deve ser [5,4,3,6,8,9].
Node* list_varredura(Node *p);

// Esta função usa a função list_varredura descrita acima para implementar
// o algoritmo de ordenação por bolha (bubblesort) numa lista encadeada.
// Complexidade: O(n^2).
// Não é preciso alocar memória nessa função.
Node* list_bubblesort(Node *p);

#endif
```

LinkedList.c

```c
#include "LinkedList.h"

struct node {
    int data;
    struct node *next;
};


// Função que cria uma lista vazia
// sem nó sentinela
Node *list_create(void) {
    return NULL;
}

// Insere um valor no início da lista
// e retorna a lista modificada.
Node *list_push_front(Node *list, int value) {
    Node* novo = (Node*) malloc(sizeof(Node));
    novo->data = value;
    novo->next = list;
    return novo;
}

// Insere um valor ao final da lista
// e retorna a lista modificada.
Node *list_push_back(Node *list, int value) {
    Node *novo = (Node *)malloc(sizeof(Node));
    novo->data = value;
    novo->next = NULL;

    if(list == NULL) {
        return novo;
    } else {
        Node *q = list;
        while(q->next != NULL) {
            q = q->next;
        }
        q->next = novo;
        return list;
    }
}

// Função que recebe um ponteiro para a cabeça da lista e
// imprime o conteúdo dos elementos na tela.
// Formato: [ a1 a2 a3 ... an ]
void list_print(Node *list) {
    printf("[ ");
    while(list != NULL) {
        printf("%d ", list->data);
        list = list->next;
    }
    printf("]\n");
}

// Função que libera todos os nós
// alocados dinamicamente
void list_free(Node *list) {
    while(list != NULL) {
        Node *aux = list->next;
        printf("list_free: nodo liberado: %d\n", list->data);
        free(list);
        list = aux;
    }
}

// Função que imprime o tamanho da lista (número de nós)
size_t list_size(Node *list) {
    size_t contador = 0;
    for(Node *atual = list; atual != NULL; atual = atual->next) {
        contador++;
    }
    return contador;
}
```

VPL

◄ [linked list] Implementar uma lista simplesmente encadeada - Parte 3

Seguir para...

Slides da aula: Variações de Listas Encadeadas ►