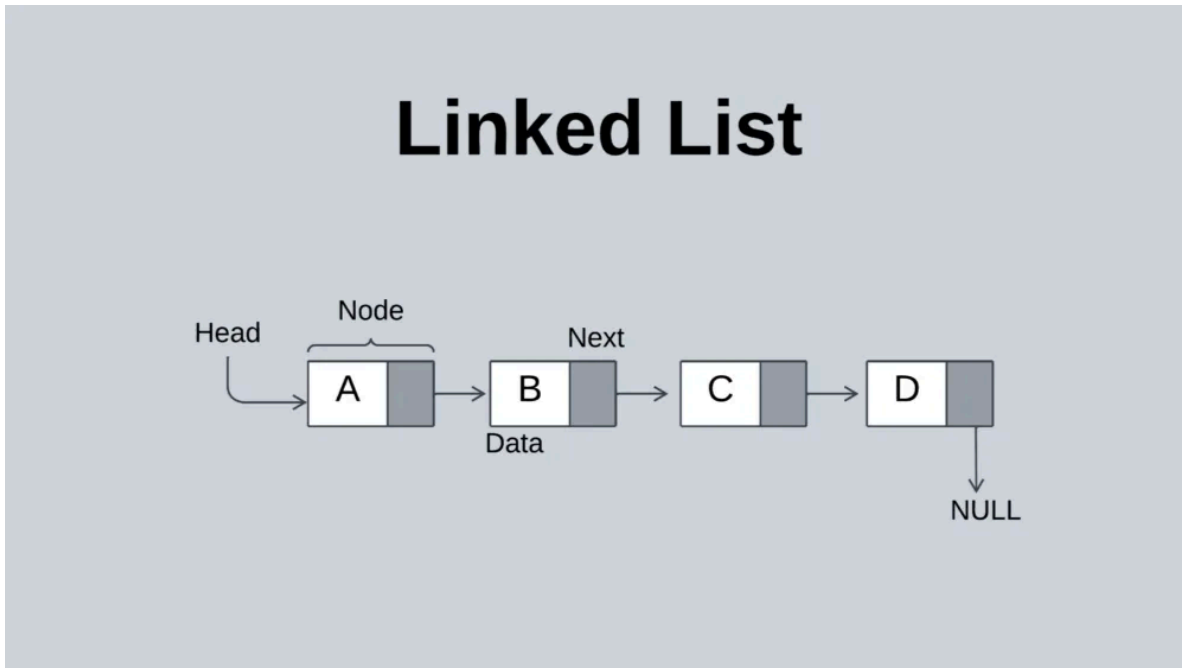
QXD0010 - ESTRUTURA DE DADOS - 01A - 2025.2

Descrição | Enviar | Editar | Visualizar envios

[linked list] Implementar uma lista simplesmente encadeada - Parte 2

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 reverte a ordem dos elementos na lista.
// Por exemplo, se você tiver a lista [10,30,40,50], então após
// chamar essa função a sua lista torna-se [50,40,30,10].
// A complexidade dessa função deve ser O(n). Além disso, faça essa
// função SEM ALOCAR NOVOS NODES. É possível fazer essa função sem
// alocar novos nós. Você precisará apenas de três ponteiros adicionais
// para te ajudar a reverter a sua lista.
// Retorna um ponteiro para a lista revertida.
Node* list_reverter(Node *p);

// Essa função recebe como parâmetro uma lista, um índice e um valor inteiro
// e insere esse inteiro na posição indicada pelo índice.
// Note que os índices estarão no intervalo [0..n-1].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_inserir_apos
// com o índice 2 e com o valor 999, essa função vai
// inserir o 999 logo após o elemento 3, resultando na lista: [1,2,3,999,4,5,6].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_inserir_apos(Node *p, size_t index, int val);
```

```
// Remove o elemento que está após o índice indicado.
// Note que os índices estarão no intervalo [0..n-2].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_remover_apos
// com o índice 4, essa função vai remover o número 6, resultando na lista: [1,2,3,4,5].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_remover_apos(Node *p, size_t index);
```

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
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio da lista k
show	mostra todas as listas na tela
addBack k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no final da lista k
insert_after k i v	insere o valor v na lista k após o índice i
remove_after k i	remove o valor após o índice i na lista k
reverse k	Inverte os elementos 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]));
        }

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

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

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

```c
        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 reverte a ordem dos elementos na lista.
// Por exemplo, se você tiver a lista [10,30,40,50], então após
// chamar essa função a sua lista torna-se [50,40,30,10].
// A complexidade dessa função deve ser O(n). Além disso, faça essa
// função SEM ALOCAR NOVOS NODES. É possível fazer essa função sem
// alocar novos nós. Você precisará apenas de três ponteiros adicionais
// para te ajudar a reverter a sua lista.
// Retorna um ponteiro para a lista revertida.
Node* list_reverter(Node *p);

// Essa função recebe como parâmetro uma lista, um índice e um valor inteiro
// e insere esse inteiro na posição indicada pelo índice.
// Note que os índices estarão no intervalo [0..n-1].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_inserir_apos
// com o índice 2 e com o valor 999, essa função vai
// inserir o 999 logo após o elemento 3, resultando na lista: [1,2,3,999,4,5,6].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_inserir_apos(Node *p, size_t index, int val);

// Remove o elemento que está após o índice indicado.
// Note que os índices estarão no intervalo [0..n-2].
// Ao final, retorna a lista modificada.
// Por exemplo, dada a lista [1,2,3,4,5,6] se voce chamar a função list_remover_apos
// com o índice 4, essa função vai remover o número 6, resultando na lista: [1,2,3,4,5].
// Se o índice for inválido a lista não é modificada. Um índice é inválido se ele for
// maior ou igual a n, ou se ele for igual a 0 e a lista estiver vazia.
Node* list_remover_apos(Node *p, size_t index);


#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 1

Seguir para...

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