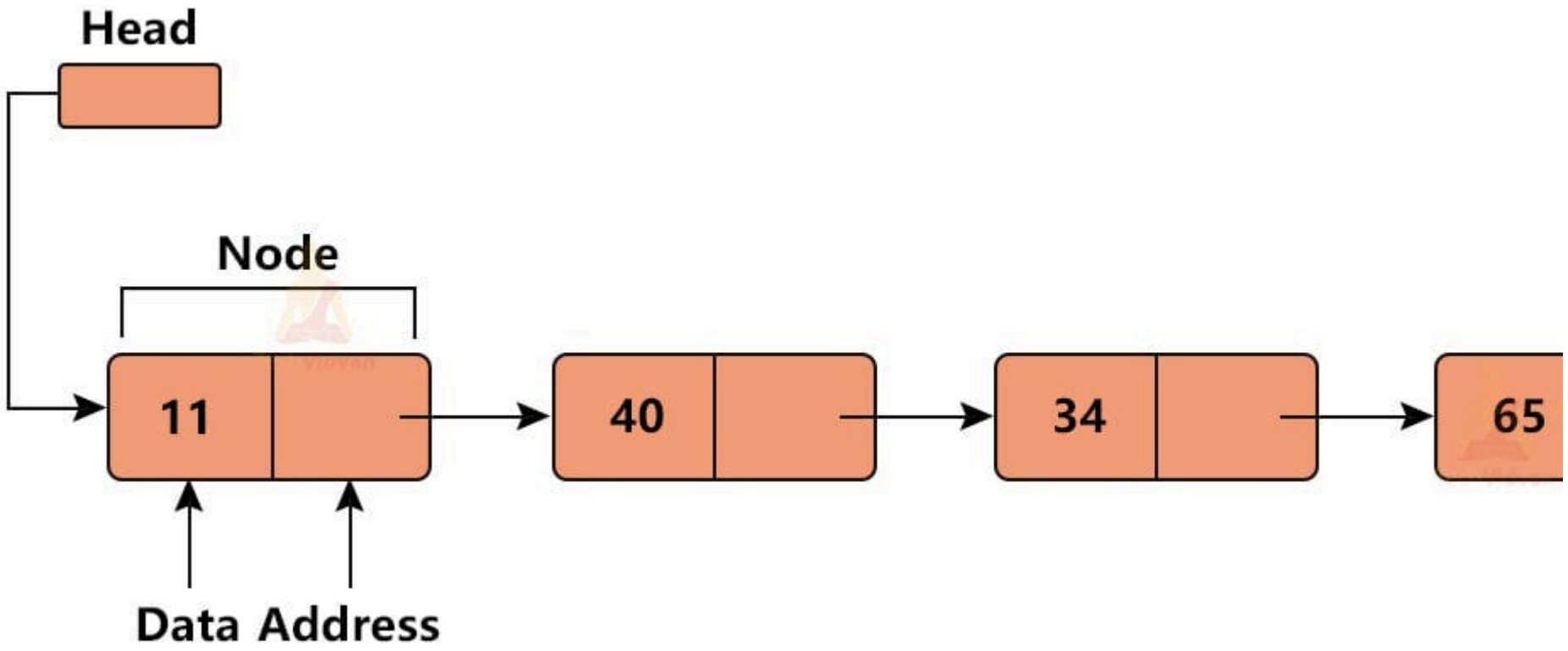
QXD0010 - ESTRUTURA DE DADOS - 01A - 2025.2

Descrição | Enviar | Editar | Visualizar envios

[linked list] Implementar uma lista simplesmente encadeada - Parte 1

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 concatena duas listas simplemente encadeadas passadas como parâmetro e
// retorna a lista resultante já concatenada.
// Exemplo se p1 = [1,2,3,4] e p2 = [44,66,77], então a lista resultante deve ser
[1,2,3,4,44,66,77].
// Se a lista p1 for vazia, deve retornar p2.
Node* list_concatenar(Node *p1, Node *p2);
// Esta função recebe como entrada uma lista e faz uma
// cópia profunda (deep copy) da lista, i.e., a lista
// resultante contém os mesmos elementos que a original só que
// elas não compartilham nós. As duas listas são independentes.
// Ou seja, essa função cria um clone da lista original e retorna o clone.
Node* list_clone(Node *p);
// Esta função remove todas as ocorrencias do valor 'val' da sua lista.
// Por exemplo, se você tiver a lista [4,1,2,4,4,3,4,4] e chamar
// essa função com o valor 4, a lista resultante sera [1,2,3].
// Não esqueça de liberar os nós que forem removidos.
```

```
// Antes de liberar o nó você deve imprimir "liberado: %d" e coloca o valor do nó liberado.
// Retorna a lista modificada.
Node* list_remove_todos(Node *p, int val);
// Esta função retorna true se as duas listas são idênticas; ou
// retorna false caso contrário.
// Duas listas são idênticas se elas têm os mesmos elementos na mesma ordem.
bool list_sao_iguais(Node* p1, Node *p2);
// Esta função retorna true se as duas listas são diferentes; ou
// retorna false caso contrário.
// Duas listas são diferentes se elas não são idênticas.
bool list_sao_diferentes(Node* p1, Node *p2);
```

Exemplo de Entrada

create
create
show
addFront 1 7 2 2 2 5 2 2 2
show
addFront 1 3 4 4 4
show
addFront 0 8 3 3 3 3 3 3 3
show
exit

Exemplo de Saída

$create
$create
$show
lista 0: [ ]
lista 1: [ ]
$addFront 1 7 2 2 2 5 2 2 2
$show
lista 0: [ ]
lista 1: [2 2 2 5 2 2 2 7]
$addFront 1 3 4 4 4
$show
lista 0: [ ]
lista 1: [4 4 4 3 2 2 2 5 2 2 2 7]
$addFront 0 8 3 3 3 3 3 3 3
$show
lista 0: [3 3 3 3 3 3 3 8]
lista 1: [4 4 4 3 2 2 2 5 2 2 2 7]
$exit
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 3
list_free: nodo liberado: 8
list_free: nodo liberado: 4
list_free: nodo liberado: 4
list_free: nodo liberado: 4
list_free: nodo liberado: 3
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 5
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 2
list_free: nodo liberado: 7

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
clone k	cria uma nova lista com os mesmos elementos da lista k já existente
show	mostra todas as listas na tela
different p q	retorna true se e somente se as listas p e q forem diferentes
remove x k	remove todos os valores x da lista k
equals p q	retorna true se e somente se as listas p e q forem iguais
reverse k	Inverte os elementos da lista k
addFront k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no inicio da lista k
concat k p	concatena a lista p ao final da lista k
addBack k a1 a2 a3 ... an	adiciona os inteiros a1, a2, ..., no final 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]));
        }

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

        // -------- concat l1 l2 --------
        else if(strcmp(token, "concat") == 0) {
            int l1 = atoi(strtok(NULL, " "));
            int l2 = atoi(strtok(NULL, " "));
            listas.data[l1] = list_concatenar(listas.data[l1], listas.data[l2]);
            listas.data[l2] = NULL;
        }

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

        // -------- equals l1 l2 --------
        else if(strcmp(token, "equals") == 0) {
            int l1 = atoi(strtok(NULL, " "));
            int l2 = atoi(strtok(NULL, " "));
            if(list_sao_iguais(listas.data[l1], listas.data[l2]))
                printf("listas iguais\n");
            else
                printf("listas diferentes\n");
        }

        // -------- different l1 l2 --------
        else if(strcmp(token, "different") == 0) {
            int l1 = atoi(strtok(NULL, " "));
            int l2 = atoi(strtok(NULL, " "));
            if(list_sao_diferentes(listas.data[l1], listas.data[l2]))
                printf("listas diferentes\n");
            else
                printf("listas iguais\n");
        }

        // -------- reverse l --------
```

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

        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 concatena duas listas simplemente encadeadas passadas como parâmetro e
// retorna a lista resultante já concatenada.
// Exemplo se p1 = [1,2,3,4] e p2 = [44,66,77], então a lista resultante deve ser [1,2,3,4,44,66,77].
// Se a lista p1 for vazia, deve retornar p2.
Node* list_concatenar(Node *p1, Node *p2);

// Esta função recebe como entrada uma lista e faz uma
// cópia profunda (deep copy) da lista, i.e., a lista
// resultante contém os mesmos elementos que a original só que
// elas não compartilham nós. As duas listas são independentes.
// Ou seja, essa função cria um clone da lista original e retorna o clone.
Node* list_clone(Node *p);


// Esta função remove todas as ocorrencias do valor 'val' da sua lista.
// Por exemplo, se você tiver a lista [4,1,2,4,4,3,4,4] e chamar
// essa função com o valor 4, a lista resultante sera [1,2,3].
// Não esqueça de liberar os nós que forem removidos.
// Antes de liberar o nó você deve imprimir "liberado: %d" e coloca o valor do nó liberado.
// Retorna a lista modificada.
Node* list_remove_todos(Node *p, int val);


// Esta função retorna true se as duas listas são idênticas; ou
// retorna false caso contrário.
// Duas listas são idênticas se elas têm os mesmos elementos na mesma ordem.
bool list_sao_iguais(Node* p1, Node *p2);


// Esta função retorna true se as duas listas são diferentes; ou
// retorna false caso contrário.
// Duas listas são diferentes se elas não são idênticas.
bool list_sao_diferentes(Node* p1, Node *p2);

#endif
```

LinkedList.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "LinkedList.h"

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


Node *list_create(void) {
    return NULL;
}

Node *list_push_front(Node *list, int value) {
    Node* novo = (Node*) malloc(sizeof(Node));
    novo->data = value;
    novo->next = list;
    return novo;
}

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;
    }
}

void list_print(Node *list) {
    printf("[ ");
    while(list != NULL) {
        printf("%d ", list->data);
        list = list->next;
    }
    printf("]\n");
}

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;
    }
}

size_t list_size(Node *list) {
    size_t contador = 0;
    for(Node *atual = list; atual != NULL; atual = atual->next) {
        contador++;
    }
    return contador;
}
```

VPL

◄ Código desenvolvido na aula do dia 25/11/2025

Seguir para...

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