Universidade Veiga de Almeida

Tecnólogo em Análise e Desenvolvimento de Sistemas

Implementação de Lista Simplesmente Encadeada em Linguagem C

Alexis Pereira Ferreira Pinto

Universidade Veiga de Almeida

Tecnólogo em Análise e Desenvolvimento de Sistemas

Implementação de Lista Simplesmente Encadeada em Linguagem C

Alexis Pereira Ferreira Pinto

Trabalho acadêmico apresentado como requisito parcial para avaliação da disciplina de Estruturas de Dados.

Resumo

Este trabalho apresenta o desenvolvimento de um programa em linguagem C capaz de realizar as operações fundamentais de manipulação de uma lista simplesmente encadeada. O programa implementa rotinas de inclusão, consulta, alteração e remoção de elementos, empregando alocação dinâmica de memória e estruturas encadeadas. A proposta integra conceitos essenciais da disciplina de Estruturas de Dados, permitindo a compreensão prática do funcionamento interno dessa estrutura dinâmica. Os resultados obtidos demonstram uma implementação funcional e organizada, adequada ao propósito de estudo e aplicação acadêmica.

Sumário

1 Introdução

As listas encadeadas constituem uma das estruturas de dados fundamentais na Ciência da Computação, amplamente utilizadas em situações que demandam flexibilidade na alocação e reorganização de elementos. Diferentemente de vetores, que utilizam posições contíguas na memória, as listas encadeadas distribuem seus elementos de forma não necessariamente sequencial, conectando-os por meio de ponteiros.

No presente trabalho, desenvolvido para a contexto acadêmico da disciplina de Estruturas de Dados, foi solicitado o projeto e implementação de um programa em linguagem C capaz de realizar operações elementares sobre uma lista simplesmente encadeada, possibilitando ao usuário inserir, consultar, alterar e remover elementos de maneira interativa.

2 Justificativa

A escolha da lista simplesmente encadeada como estrutura de dados para este trabalho se justifica por sua adequação a cenários onde a quantidade de elementos é variável e imprevisível. Essa característica torna as listas encadeadas ideais para aplicações que exigem inserções e remoções frequentes, evitando o custo adicional de realocação presente em estruturas estáticas ou semidinâmicas.

Além disso, o uso de ponteiros e alocação dinâmica em C oferece ao programador controle total sobre o gerenciamento de memória, permitindo otimizações e aprofundamento no entendimento das operações internas da estrutura. Assim, o desenvolvimento deste programa contribui não apenas para o atendimento do enunciado, mas também para a formação técnica aprofundada em manipulação de dados dinâmicos.

3 Metodologia

A construção do programa seguiu as seguintes etapas metodológicas:

1. **Definição da Estrutura do Nó**: Foi criada uma estrutura contendo um campo de dado (string alocada dinamicamente) e um ponteiro para o próximo nó.

2. **Implementação das Funções**: Cada operação solicitada no enunciado foi desenvolvida como uma função independente, garantindo modularidade:

    - Inserção de um novo elemento ao final da lista.
    - Consulta baseada em posição.
    - Alteração a partir da busca textual.
    - Remoção de um nó, com realinhamento de ponteiros.

3. **Gerenciamento de Memória**: Todas as strings manipuladas pelo programa são alocadas dinamicamente, bem como os nós da lista. Após remoções, a memória é devidamente liberada.

4. **Construção do Menu Interativo**: Foi criada uma interface textual que permite ao usuário realizar as operações de forma sequencial, controlando o fluxo do programa.

O uso de boas práticas, como validação de ponteiros, modularização e comentários explicativos, garantiu a clareza de sua estrutura.

4 Conclusão

O presente trabalho cumpriu integralmente o objetivo proposto no enunciado, resultando em um programa funcional capaz de manipular uma lista simplesmente encadeada em linguagem C. A implementação contemplou todas as operações essenciais, inclusão, consulta, alteração e remoção, demonstrando o domínio de conceitos de estruturas de dados dinâmicas e manipulação de ponteiros.

A realização deste projeto reforça a importância das listas encadeadas como ferramenta didática e prática, permitindo ao programador compreender profundamente os mecanismos internos de armazenamento dinâmico. Além disso, a solução desenvolvida oferece base sólida para futuras expansões, como a implementação de listas duplamente encadeadas, ordenadas ou circulares.

Referências

1 SZWARCFITER, J. L.; MARKENZON, L. *Estruturas de dados e seus algoritmos*. Rio de Janeiro: Livros Técnicos e Científicos, 1994.

2 CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. *Introdução a Estruturas de Dados – com técnicas de programação em C*. Rio de Janeiro: GEN LTC, 2016.

3 PINHEIRO, Francisco de Assis Cartaxo. *Elementos de programação em C*. Porto Alegre: Bookman, 2012.

A Código-Fonte Completo

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/*
 * Estrutura básica do nó de uma lista simplesmente encadeada.
 * Cada nó contém:
 * - Um ponteiro para um dado armazenado dinamicamente (string)
 * - Um ponteiro para o nó seguinte
 */
typedef struct No {
    char *dado;         // Armazena o dado do nó
    struct No *prox;    // Ponteiro para o próximo nó da lista
} No;

// Ponteiro global que representa o início da lista encadeada
No *inicio = NULL;

/*
 * Função: lerString
 * -----------------
 * Lê uma string de tamanho variável do teclado, utilizando um buffer temporário
 * e posteriormente alocando memória exata para armazenar o texto digitado.
 *
 * Retorna:
 *  Um ponteiro para uma string alocada dinamicamente.
 */
char* lerString() {
    char buffer[1024];

    fgets(buffer, 1024, stdin);
```

```
32      buffer[strcspn(buffer, "\n")] = '\0';  // remove o '\n'
33
34      char *str = (char*) malloc(strlen(buffer) + 1);
35
36      if (str == NULL) {
37          printf("Erro ao alocar memória.\n");
38          exit(1);
39      }
40
41      strcpy(str, buffer);
42      return str;
43  }
44
45  /*
46   * Função: criarNo
47   * ----------------
48   * Cria um novo nó para a lista encadeada.
49   *
50   * Parâmetro:
51   *  valor - string já alocada dinamicamente.
52   *
53   * Retorna:
54   *  Ponteiro para o novo nó criado.
55   */
56  No* criarNo(char *valor) {
57      No *novo = (No*) malloc(sizeof(No));
58
59      if (novo == NULL) {
60          printf("Erro ao alocar memória para o nó.\n");
61          exit(1);
62      }
63
64      novo->dado = valor;
65      novo->prox = NULL;
66
67      return novo;
68  }
69
70  /*
71   * Função: inserir
72   * ----------------
73   * Insere um novo nó no final da lista encadeada.
74   *
75   * Procedimento:
76   *  - Caso a lista esteja vazia, o novo nó se torna o início.
77   *  - Caso contrário, percorre até o último nó e o conecta ao novo.
78   */
```

```c
void inserir(char *valor) {
    No *novo = criarNo(valor);

    if (inicio == NULL) {    // lista vazia
        inicio = novo;
        return;
    }

    No *aux = inicio;
    while (aux->prox != NULL)
        aux = aux->prox;

    aux->prox = novo;
}

/*
 * Função: listar
 * --------------
 * Percorre a lista encadeada do início ao fim,
 * exibindo os dados de cada nó.
 */
void listar() {
    if (inicio == NULL) {
        printf("Lista vazia.\n");
        return;
    }

    No *aux = inicio;
    printf("Elementos da lista:\n");
    while (aux != NULL) {
        printf("- %s\n", aux->dado);
        aux = aux->prox;
    }
}

/*
 * Função: consultar
 * -----------------
 * Consulta um elemento da lista baseado em sua posição.
 *
 * Parâmetro:
 *  pos - índice do elemento na lista (0 = primeiro).
 *
 * Retorna:
 *  O dado encontrado ou NULL se a posição for inválida.
 */
char* consultar(int pos) {
```

```c
    if (pos < 0)      // posição negativa não é válida
        return NULL;

    No *aux = inicio;
    int i = 0;

    while (aux != NULL) {
        if (i == pos)
            return aux->dado;

        aux = aux->prox;
        i++;
    }

    return NULL;      // posição além do tamanho da lista
}

/*
 * Função: alterar
 * ----------------
 * Altera o conteúdo de um nó buscando pelo valor antigo.
 *
 * Parâmetros:
 *  antigo - valor que será buscado na lista.
 *  novoValor - novo texto que substituirá o anterior.
 *
 * Retorna:
 *  1 se a alteração foi realizada; 0 caso contrário.
 */
int alterar(char *antigo, char *novoValor) {
    No *aux = inicio;

    while (aux != NULL) {
        if (strcmp(aux->dado, antigo) == 0) {
            free(aux->dado);       // libera o texto antigo
            aux->dado = novoValor;  // substitui pelo novo
            return 1;
        }
        aux = aux->prox;
    }

    return 0;  // dado não encontrado
}

/*
 * Função: removerElemento
 * -----------------------
```

```
 * Remove um nó da lista encadeada baseado no seu conteúdo.
 *
 * Parâmetro:
 *  valor - string que será procurada na lista.
 *
 * Procedimento:
 *  - Ajusta os ponteiros para desconectar o nó removido.
 *  - Libera a memória alocada para o dado e para o nó.
 *
 * Retorna:
 *  1 se removido; 0 se não encontrado.
 */
int removerElemento(char *valor) {
    if (inicio == NULL)   // lista vazia
        return 0;

    No *aux = inicio;
    No *anterior = NULL;

    while (aux != NULL && strcmp(aux->dado, valor) != 0) {
        anterior = aux;
        aux = aux->prox;
    }

    if (aux == NULL)      // não encontrou
        return 0;

    if (anterior == NULL)     // remoção do primeiro nó
        inicio = aux->prox;
    else
        anterior->prox = aux->prox;

    free(aux->dado);
    free(aux);

    return 1;
}

/*
 * Função: menu
 * ------------
 * Exibe as opções do usuário.
 */
void menu() {
    printf("\n=====␣LISTA␣ENCADEADA␣=====\n");
    printf("1␣-␣Inserir\n");
    printf("2␣-␣Listar\n");
```

```
220     printf("3 - Consultar por posição\n");
221     printf("4 - Alterar\n");
222     printf("5 - Remover\n");
223     printf("0 - Sair\n");
224     printf("Escolha: ");
225 }
226 
227 /*
228  * Função principal
229  * -----------------
230  * Gerencia a interação com o usuário e chama as funções da lista.
231  */
232 int main() {
233     int op;
234 
235     do {
236         menu();
237         scanf("%d", &op);
238         getchar();  // consome o '\n' deixado pelo scanf
239 
240         char *valor, *novo;
241 
242         switch (op) {
243 
244             case 1:
245                 printf("Insira os dados: ");
246                 valor = lerString();
247                 inserir(valor);
248                 break;
249 
250             case 2:
251                 listar();
252                 break;
253 
254             case 3: {
255                 printf("Posição da lista a consultar: ");
256                 int pos;
257                 scanf("%d", &pos);
258                 getchar();
259 
260                 char *resultado = consultar(pos);
261 
262                 if (resultado)
263                     printf("Na posição %d está: %s\n", pos, resultado);
264                 else
265                     printf("Posição inválida.\n");
266                 break;
```

```
267             }
268
269             case 4:
270                 printf("Texto atual: ");
271                 valor = lerString();
272                 printf("Novo texto: ");
273                 novo = lerString();
274
275                 if (alterar(valor, novo))
276                     printf("Alterado com sucesso!\n");
277                 else {
278                     printf("Não encontrado.\n");
279                     free(novo);
280                 }
281
282                 free(valor);
283                 break;
284
285             case 5:
286                 printf("Texto a remover: ");
287                 valor = lerString();
288
289                 if (removerElemento(valor))
290                     printf("Removido.\n");
291                 else
292                     printf("Não encontrado.\n");
293
294                 free(valor);
295                 break;
296
297             case 0:
298                 printf("Saindo...\n");
299                 break;
300
301             default:
302                 printf("Opção inválida!\n");
303         }
304
305     } while (op != 0);
306
307     return 0;
308 }
```