Rafael Kazuhito Vidal Beltrão - ADS Turma C

Atividade realizada para a matéria de ESTRUTURA DE DADOS

Repositório com as atividades
[https://github.com/rkazuhito/Faculdade/tree/main/2o%20semestre/estrutura%20de%20dados/atividadeLista]

LISTA COMENTADA

```
#include <stdio.h> //biblioteca para entrada e saída

#include <stdlib.h> //biblioteca para alocação


struct Node{ //estrutura de nó

int num; //cada nó contem um num

struct Node *prox; //ponteiro para o proximo nó

};

typedef struct Node node; //definição do nó - node


int tam; //var global para o tamanho da lista


void inicia(node *LISTA);  //Esses são os protótipos das funções que serão definidas posteriormente no código. Eles fornecem uma descrição dos tipos de argumentos e valores de retorno de cada função.

int menu(void);

void opcao(node *LISTA, int op);

node *criaNo();

void insereFim(node *LISTA);

void insereInicio(node *LISTA);

void exibe(node *LISTA);

void libera(node *LISTA);

void insere (node *LISTA);

node *retiraInicio(node *LISTA);

node *retiraFim(node *LISTA);

node *retira(node *LISTA);
```

```
int main(void) //função principal de execução do programa
{
node *LISTA = (node *) malloc(sizeof(node));  //alocacao do nó inicial
if(!LISTA){ //verifica se a alocacao foi possivel, caso n seja, entra no if
  printf("Sem memoria disponivel!\n");
  exit(1);
}else{ //se for alocado, inicia a lista
inicia(LISTA); //funcao para comecar a lista
int opt; //variavel de opcao

do{ //loop de giro principal do programa
  opt=menu();
  opcao(LISTA,opt);
}while(opt);

free(LISTA); //libera a memória alocada
return 0;
}
}

void inicia(node *LISTA) //inicia a lista
{
LISTA->prox = NULL; //indica o proximo nó como vazio, lista esta vazia
tam=0; //tamanho zeo = lista vazia
}

int menu(void) //funcao para fazer o menu
{
int opt;
```

```
printf("Escolha a opcao\n"); //menu sendo exibido na tela
printf("0. Sair\n");
printf("1. Zerar lista\n");
printf("2. Exibir lista\n");
printf("3. Adicionar node no inicio\n");
printf("4. Adicionar node no final\n");
printf("5. Escolher onde inserir\n");
printf("6. Retirar do inicio\n");
printf("7. Retirar do fim\n");
printf("8. Escolher de onde tirar\n");
printf("Opcao: ");
scanf("%d", &opt); //le a opcao escolhida pelo usuario

return opt; // Retorna a opção escolhida pelo usuário
}

void opcao(node *LISTA, int op) { // Função para executar a operação escolhida pelo usuário
   node *tmp;
   switch(op) { // Realiza uma operação dependendo da opção escolhida pelo usuário
      case 0: // Caso a opção seja 0 (Sair), libera a memória e encerra o programa
         libera(LISTA);
         break;
      case 1: // Caso a opção seja 1 (Zerar lista), libera a memória e reinicia a lista
         libera(LISTA);
         inicia(LISTA);
         break;
      case 2:// Caso a opção seja 2 (Exibir lista), exibe os elementos da lista
         exibe(LISTA);
         break;
      case 3: // Caso a opção seja 3 (Adicionar nó no início), insere um nó no início da lista
```

```
            insereInicio(LISTA);
            break;
        case 4: // Caso a opção seja 4 (Adicionar nó no final), insere um nó no final da lista
            insereFim(LISTA);
            break;
        case 5: // Caso a opção seja 5 (Escolher onde inserir), insere um nó em uma posição específica
            insere(LISTA);
            break;
        case 6: // Caso a opção seja 6 (Retirar do início), retira um nó do início da lista
            tmp = retiraInicio(LISTA);
            printf("Retirado: %3d\n\n", tmp->num);
            break;
        case 7:  // Caso a opção seja 7 (Retirar do fim), retira um nó do fim da lista
            tmp = retiraFim(LISTA);
            printf("Retirado: %3d\n\n", tmp->num);
            break;
        case 8: // Caso a opção seja 8 (Escolher de onde tirar), retira um nó de uma posição específica
            tmp = retira(LISTA);
            printf("Retirado: %3d\n\n", tmp->num);
            break;
        default: // Se a opção for inválida, exibe uma mensagem de erro
            printf("Comando invalido\n\n");
    }
}

// Função para verificar se a lista está vazia
int vazia(node *LISTA) {
    // Verifica se o próximo nó do nó inicial é NULL
    if (LISTA->prox == NULL)
        return 1; // Retorna 1 se a lista estiver vazia
    else
```

```
        return 0; // Retorna 0 se a lista não estiver vazia
}
// Função para alocar dinamicamente um novo nó
node *aloca() {
   // Declaração de um ponteiro para um novo nó
   node *novo = (node *) malloc(sizeof(node));
   // Verifica se a alocação de memória foi bem-sucedida
   if (!novo) {
      // Se não houver memória disponível, exibe uma mensagem de erro e encerra o programa
      printf("Sem memoria disponivel!\n");
      exit(1);
   } else {
      // Se a alocação for bem-sucedida, solicita ao usuário um número para armazenar no novo nó
      printf("Novo elemento: ");
      scanf("%d", &novo->num);
      // Retorna o ponteiro para o novo nó alocado
      return novo;
   }
}

// Função para inserir um nó no final da lista
void insereFim(node *LISTA) {
   // Aloca um novo nó
   node *novo = aloca();
   // Define o próximo do novo nó como NULL, já que ele será o último na lista
   novo->prox = NULL;

   // Verifica se a lista está vazia
   if (vazia(LISTA))
      // Se a lista estiver vazia, o novo nó se torna o primeiro da lista
      LISTA->prox = novo;
```

```
    else {
        // Caso contrário, percorre a lista até encontrar o último nó
        node *tmp = LISTA->prox;
        while (tmp->prox != NULL)
            tmp = tmp->prox;

        // Insere o novo nó após o último nó encontrado
        tmp->prox = novo;
    }
    // Incrementa o tamanho da lista
    tam++;
}

// Função para inserir um nó no início da lista
void insereInicio(node *LISTA) {
    // Aloca um novo nó
    node *novo = aloca();
    // Guarda o endereço do primeiro nó atual
    node *oldHead = LISTA->prox;

    // Faz o próximo do novo nó apontar para o antigo primeiro nó
    LISTA->prox = novo;
    // Faz o próximo do novo primeiro nó apontar para o antigo primeiro nó
    novo->prox = oldHead;

    // Incrementa o tamanho da lista
    tam++;
}



// Função para exibir os elementos da lista
```

```
void exibe(node *LISTA) {
   // Limpa a tela do console
   system("clear");
   // Verifica se a lista está vazia
   if (vazia(LISTA)) {
      // Se estiver vazia, exibe uma mensagem indicando que a lista está vazia
      printf("Lista vazia!\n\n");
      return;
   }

   // Declara um ponteiro temporário para percorrer a lista
   node *tmp;
   // Inicializa o ponteiro temporário com o primeiro nó da lista
   tmp = LISTA->prox;
   // Exibe os elementos da lista e suas posições
   printf("Lista:");
   while (tmp != NULL) {
      printf("%5d", tmp->num);
      tmp = tmp->prox;
   }
   printf("\n       ");

   // Exibe setas indicando a ordem dos elementos
   int count;
   for (count = 0; count < tam; count++)
      printf("  ^  ");
   printf("\nOrdem:");
   for (count = 0; count < tam; count++)
      printf("%5d", count + 1);

   printf("\n\n");
```

```
}


// Função para liberar a memória alocada para os nós da lista
void libera(node *LISTA) {
   // Verifica se a lista não está vazia
   if (!vazia(LISTA)) {
      node *proxNode, *atual;
      // Inicializa o ponteiro atual com o primeiro nó da lista
      atual = LISTA->prox;
      // Enquanto houver nós na lista
      while (atual != NULL) {
         // Guarda o próximo nó da lista
         proxNode = atual->prox;
         // Libera a memória alocada para o nó atual
         free(atual);
         // Atualiza o ponteiro atual para apontar para o próximo nó
         atual = proxNode;
      }
   }
}


// Função para inserir um nó em uma posição específica da lista
void insere(node *LISTA) {
   int pos, count;
   // Solicita ao usuário a posição onde deseja inserir o nó
   printf("Em que posicao, [de 1 ate %d] voce deseja inserir: ", tam);
   scanf("%d", &pos);

   // Verifica se a posição fornecida é válida
```

```
    if (pos > 0 && pos <= tam) {
        // Se a posição for o início da lista, insere no início
        if (pos == 1)
            insereInicio(LISTA);
        else {
            node *atual = LISTA->prox, *anterior = LISTA;
            // Aloca um novo nó
            node *novo = aloca();

            // Percorre a lista até encontrar a posição desejada
            for (count = 1; count < pos; count++) {
                anterior = atual;
                atual = atual->prox;
            }

            // Insere o novo nó na posição desejada
            anterior->prox = novo;
            novo->prox = atual;
            // Incrementa o tamanho da lista
            tam++;
        }
    } else {
        // Se a posição fornecida for inválida, exibe uma mensagem de erro
        printf("Elemento invalido\n\n");
    }
}
// Função para retirar o primeiro nó da lista
node *retiraInicio(node *LISTA) {
    // Verifica se a lista está vazia
    if (LISTA->prox == NULL) {
        // Se estiver vazia, exibe uma mensagem informando que a lista já está vazia
```

```
        printf("Lista ja esta vazia\n");
        return NULL;
    } else {
        // Se não estiver vazia, guarda o endereço do primeiro nó
        node *tmp = LISTA->prox;
        // Atualiza o ponteiro do nó inicial para apontar para o próximo nó
        LISTA->prox = tmp->prox;
        // Decrementa o tamanho da lista
        tam--;
        // Retorna o nó removido
        return tmp;
    }
}
// Função para retirar o último nó da lista
node *retiraFim(node *LISTA) {
    // Verifica se a lista está vazia
    if (LISTA->prox == NULL) {
        // Se estiver vazia, exibe uma mensagem informando que a lista já está vazia
        printf("Lista ja vazia\n\n");
        return NULL;
    } else {
        // Declara dois ponteiros para percorrer a lista
        node *ultimo = LISTA->prox, *penultimo = LISTA;

        // Percorre a lista até encontrar o último nó
        while (ultimo->prox != NULL) {
            penultimo = ultimo;
            ultimo = ultimo->prox;
        }

        // Atualiza o ponteiro do nó anterior ao último para apontar para NULL
```

```
        penultimo->prox = NULL;
        // Decrementa o tamanho da lista
        tam--;
        // Retorna o último nó removido
        return ultimo;
    }
}

// Função para retirar um nó de uma posição específica da lista
node *retira(node *LISTA) {
    int opt, count;
    // Solicita ao usuário a posição do nó que deseja retirar
    printf("Que posicao, [de 1 ate %d] voce deseja retirar: ", tam);
    scanf("%d", &opt);

    // Verifica se a posição fornecida é válida
    if (opt > 0 && opt <= tam) {
        // Se a posição for o início da lista, remove o primeiro nó
        if (opt == 1)
            return retiraInicio(LISTA);
        else {
            node *atual = LISTA->prox, *anterior = LISTA;

            // Percorre a lista até encontrar a posição desejada
            for (count = 1; count < opt; count++) {
                anterior = atual;
                atual = atual->prox;
            }

            // Atualiza o ponteiro do nó anterior ao nó a ser retirado para apontar para o próximo nó após o nó a ser retirado
```

```
            anterior->prox = atual->prox;
            // Decrementa o tamanho da lista
            tam--;
            // Retorna o nó retirado
            return atual;
        }
    } else {
        // Se a posição fornecida for inválida, exibe uma mensagem de erro e retorna NULL
        printf("Elemento invalido\n\n");
        return NULL;
    }
}
```

LISTA ADAPTADA PARA CHAR

```
#include <iostream>
using namespace std;

struct Node {
    char data;
    Node* next;

    Node(char data) : data(data), next(nullptr) {}
};

class LinkedList {
private:
    Node* head;
public:
    LinkedList() : head(nullptr) {}
```

```
void zerarLista() {
    while (head != nullptr) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    cout << "Lista zerada." << endl;
}

void exibirLista() {
    if (head == nullptr) {
        cout << "Lista vazia." << endl;
        return;
    }
    Node* current = head;
    cout << "Lista: ";
    while (current != nullptr) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void adicionarInicio(char value) {
    Node* newNode = new Node(value);
    newNode->next = head;
    head = newNode;
    cout << "Node adicionado no inicio." << endl;
}

void adicionarFinal(char value) {
```

```
    Node* newNode = new Node(value);
    if (head == nullptr) {
        head = newNode;
        cout << "Node adicionado no final." << endl;
        return;
    }
    Node* current = head;
    while (current->next != nullptr) {
        current = current->next;
    }
    current->next = newNode;
    cout << "Node adicionado no final." << endl;
}

void adicionarPosicao(char value) {
    cout << "Função de adicionar em posição não implementada." << endl;
}

void retirarInicio() {
    if (head == nullptr) {
        cout << "Lista vazia, nada a retirar." << endl;
        return;
    }
    Node* temp = head;
    head = head->next;
    delete temp;
    cout << "Node retirado do inicio." << endl;
}

void retirarFinal() {
    if (head == nullptr) {
```

```
        cout << "Lista vazia, nada a retirar." << endl;
        return;
    }
    if (head->next == nullptr) {
        delete head;
        head = nullptr;
        cout << "Node retirado do final." << endl;
        return;
    }
    Node* current = head;
    while (current->next->next != nullptr) {
        current = current->next;
    }
    delete current->next;
    current->next = nullptr;
    cout << "Node retirado do final." << endl;
}

void retirarPosicao() {
    cout << "Função de retirar em posição não implementada." << endl;
}

void mostrarMenu() {
    char opt;
    do {
        cout << "Escolha a opcao\n";
        cout << "0. Sair\n";
        cout << "1. Zerar lista\n";
        cout << "2. Exibir lista\n";
        cout << "3. Adicionar node no inicio\n";
        cout << "4. Adicionar node no final\n";
```

```
cout << "5. Escolher onde inserir\n";
cout << "6. Retirar do inicio\n";
cout << "7. Retirar do fim\n";
cout << "8. Escolher de onde tirar\n";
cout << "Opcao: ";
cin >> opt;

switch(opt) {
   case '0':
      cout << "Saindo..." << endl;
      break;
   case '1':
      zerarLista();
      break;
   case '2':
      exibirLista();
      break;
   case '3': {
      char value;
      cout << "Digite o valor a ser adicionado no inicio: ";
      cin >> value;
      adicionarInicio(value);
      break;
   }
   case '4': {
      char value;
      cout << "Digite o valor a ser adicionado no final: ";
      cin >> value;
      adicionarFinal(value);
      break;
   }
```

```
            case '5': {
                char value;
                cout << "Digite o valor a ser adicionado na posicao: ";
                cin >> value;
                adicionarPosicao(value);
                break;
            }
            case '6':
                retirarInicio();
                break;
            case '7':
                retirarFinal();
                break;
            case '8':
                retirarPosicao();
                break;
            default:
                cout << "Opcao invalida!" << endl;
        }
    } while(opt != '0');
  }
};

int main() {
  LinkedList lista;
  lista.mostrarMenu();
  return 0;
}
```

O código foi executado em um compilador online

Output

```
/tmp/74VneIm5G4.o
Escolha a opcao
0. Sair
1. Zerar lista
2. Exibir lista
3. Adicionar node no inicio
4. Adicionar node no final
5. Escolher onde inserir
6. Retirar do inicio
7. Retirar do fim
8. Escolher de onde tirar
Opcao: 3
Digite o valor a ser adicionado no inicio: a
Node adicionado no inicio.
```

```
Escolha a opcao
0. Sair
1. Zerar lista
2. Exibir lista
3. Adicionar node no inicio
4. Adicionar node no final
5. Escolher onde inserir
6. Retirar do inicio
7. Retirar do fim
8. Escolher de onde tirar
Opcao: 3
Digite o valor a ser adicionado no inicio: b
Node adicionado no inicio.
```

```
Escolha a opcao
0. Sair
1. Zerar lista
2. Exibir lista
3. Adicionar node no inicio
4. Adicionar node no final
5. Escolher onde inserir
6. Retirar do inicio
7. Retirar do fim
8. Escolher de onde tirar
Opcao: 2
Lista: b a
```

```
Escolha a opcao
0. Sair
1. Zerar lista
2. Exibir lista
3. Adicionar node no inicio
4. Adicionar node no final
5. Escolher onde inserir
6. Retirar do inicio
7. Retirar do fim
8. Escolher de onde tirar
Opcao: 4
Digite o valor a ser adicionado no final: c
Node adicionado no final.
```

```
Opcao: 2
Lista: b a c
Escolha a opcao
```

```
Opcao: 7
Node retirado do final.
```

```
Opcao: 2
Lista: b a
```

```
Opcao: 1
Lista zerada.
```

```
Opcao: 2
Lista vazia.
```