UNIVERSIDADE FEDERAL DE UBERLÂNDIA - UFU
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO - BACHARELADO

Arthur Borges Martins (12411BCC063)
Gustavo Luís de Siqueira Nascimento (12411BCC017)
Estevão Polom Gomes (12411BCC031)

TRABALHO FINAL DE AED: SISTEMA DE RECEITAS E INGREDIENTES

UBERLÂNDIA - MG
2025

SUMÁRIO

1. INTRODUÇÃO

Nosso programa foi produzido para lidar com um Sistema de Receitas e Ingredientes, um livro de receitas onde cada receita terá sua lista de ingredientes. Definimos arquivos como *livro.c* e *livro.h* e *receita.h* e *receita.c* para organizar. Sua estrutura foi pensada a partir de uma lista simplesmente encadeada de receitas, no qual essas têm uma lista duplamente encadeada de ingredientes.

2. ARQUIVOS PRINCIPAIS

Os arquivos principais são:

- main.c: Contém a função main e a lógica principal do programa, incluindo a interação com o usuário através de menus.
- livro.h: Define a interface para manipulação do livro de receitas.
- livro.c: Implementa as funções para manipulação do livro de receitas.
- receita.h: Define a interface para manipulação de receitas e ingredientes.
- receita.c: Implementa as funções para manipulação de receitas e ingredientes.
- dados.h: Define a função para cadastrar dados de teste.
- dados.c: Implementa a função para cadastrar dados de teste.

3. ESTRUTURAS DE DADOS FUNDAMENTAIS

As estruturas de dados fundamentais são:

- struct Ingrediente: Representa nosso ingrediente, possui informações como seu nome, sua quantidade, medida e se é um ingrediente essencial.
- struct NoIngrediente: Um nó de uma lista duplamente encadeada, é quem organiza os ingredientes de uma receita. Cada nó possui um Ingrediente e ponteiros para o nó próximo e anterior.
- struct Receita: Representa nossa receita, possui seu nome, uma lista encadeada NoIngrediente, se foi favoritada e um ponteiro para a próxima receita no Livro.
- struct Livro: Representa nosso livro de receitas, contendo o nome do livro e uma lista encadeada de Receita.

4. FUNÇÕES PARA MANIPULAÇÃO

Nosso problema foi resolvido com funções para a criação e manipulação de uma lista de Receita e Livro. Para controlar os ingredientes, as funções definidas no arquivo *receita.h* foram:

- insereIngrediente(): Insere um novo ingrediente na nossa lista de ingredientes para uma receita.
- removeIngrediente(): Remove um ingrediente.
- trocar(): Troca a posição de dois ingredientes na lista.
- substituir(): Substitui a posição de um ingrediente na lista por outro.
- mostrarReceita(): Mostra os ingredientes de uma receita.
- mostraEssenciais(): Exibe apenas os ingredientes essenciais.

Enquanto no arquivo *livro.h* foram:

- insereReceita(): Insere uma nova receita ao livro.
- removeReceita(): Remove uma receita.

- recebeReceita(): Acessa uma receita escolhida no livro.
- mostraLivro(): Mostra todas as receitas.
- mostraFavoritas(): Mostra somente aquelas marcadas como favoritas.

Nossa função main() serve como uma demonstração do uso, adicionando ingredientes, criando receitas, encaixando-as em nosso livro e realizando operações como mostrar, remover e acessar as receitas. Void menu2(Receita receita) exibe o menu de opções para manipulação de uma receita específica. O arquivo dados.h define a função cadastrarDados(). O arquivo dados.c implementa a função cadastrarDados().

5. DOCUMENTAÇÃO DO CÓDIGO

5.1 RECEITA

Começando pelo arquivo *receita.h,* ele quem define as estruturas de dados e os protótipos das funções para manipular as receitas e seus ingredientes.

struct Ingrediente:

- int codigo: Identificador único para o ingrediente dentro da receita.
- char nomeIngrediente[50]: Nome do ingrediente.
- int quantidade: Quantidade do ingrediente.
- char medida[50]: A unidade de medida do ingrediente.
- int essencial: Indica se o ingrediente é ou não (0 ou 1) essencial para receita.

struct NoIngrediente:

- Ingrediente ingrediente: A struct Ingrediente armazenada nesse nó.
- struct noIngrediente *proximo: Ponteiro para o próximo nó na nossa lista de ingredientes.
- struct noIngrediente *anterior: Ponteiro para o nó anterior na lista de ingredientes.

struct descritorReceita:

- char nomeReceita[50]: Nome da receita.
- NoIngrediente inicio: Ponteiro para o primeiro nó da lista de ingredientes.
- NoIngrediente final: Ponteiro para o último nó da lista de ingredientes.
- struct descritorReceita* próxima: Ponteiro para próxima receita na lista do livro de receitas.
- int favorita: Indica se a receita é ou não favorita (0 ou 1).
- int quantidadeIngredientes: Quantidade de ingredientes na receita.

Protótipo das funções:

- Receita criaReceita(char nome[50], int favorita);
- int insereIngrediente(Ingrediente ingredienteRecebido, Receita receita);
- void mostraReceita(Receita receita);
- int removeIngrediente(int id, Receita receita);
- int trocar(Receita receita, int cod1, int cod2);
- void mostraEssenciais(Receita receita);
- int substituir(Receita receita, int cod, Ingrediente ingrediente);
- void desalocaReceita(Receita receita);

No arquivo *receita.c* temos as funções declaras em *receita.h*:

Receita criaReceita(char nome[50], int favorita) tem a função de criar uma nova receita, alocando memória para struct Receita e inicializando seus campos.

- Receita receita = malloc(sizeof(struct descritorReceita)): Aloca memória dinamicamente para uma nova Receita.
- strcpy(receita->nomeReceita, nome): Copia o nome da receita para campo nomeReceita.
- receita->inicio = NULL: Define o inicio e o final como NULL para indicar que a lista está vazia.
- receita->quantidadeIngredientes = 0: Define a quantidadeIngredientes como 0.
- receita->favorita = favorita: Define se a receita é favorita.
- receita->proxima = NULL: Define o ponteiro próxima como NULL.
- return receita: Retorna o ponteiro para nova Receita criada.

int insereIngrediente(Ingrediente ingredienteRecebido, Receita receita) tem a função de inserir um ingrediente na lista de ingredientes de uma receita.

- if(!receita) { return -1; }: Verifica se a receita existe ou é válida.
- NoIngrediente no = malloc(sizeof(struct noIngrediente)): Aloca memória para um novo NoIngrediente.
- no->ingrediente = ingredienteRecebido: Copia os dados do ingredienteRecebido para o novo NoIngrediente.
- if(receita->quantidadeIngredientes == 0) { receita->inicio = no; receita->final = no; no->anterior = NULL; no->proximo = NULL; receita->quantidadeIngredientes++; }: Se a lista de ingredientes estiver vazia, definimos o novo nó como o inicio e o final da nossa lista. Define os ponteiros anterior e proximo do nó como NULL. Incrementa o quantidadeIngredientes da receita.
- else { receita->final->proximo = no; no->proximo = NULL; no->anterior = receita->final; receita->final = no; receita->quantidadeIngredientes++; }: Se ela não estiver vazia, definimos o proximo do nó final atual para o novo nó e o anterior do novo nó como o final atual. Define o proximo do novo nó como NULL e atualiza o final da receita para o novo nó. Incrementa o quantidadeIngredientes da receita.
- no->ingrediente.codigo = receita->quantidadeIngredientes: Atribui um código ao ingrediente.
- return 0: Retorna 0 para indicar que a inserção deu certo.

int removeIngrediente(int id, Receita receita) tem a função de remover um ingrediente da lista de ingredientes da receita, dado seu id.

- if(!receita) { return -1; }: Verifica se a receita existe ou é válida.
- if(receita->quantidadeIngredientes == 0) { return -2; }: Verifica se a lista não está vazia.
- if(id > receita->quantidadeIngredientes) { return -3; }: Verifica se o id é válido.
- NoIngrediente temp = receita->inicio: Cria um ponteiro temp que aponta para o primeiro nó da lista de ingrediente da receita.
- while(temp != NULL && temp->ingrediente.codigo != id) { temp = temp->proximo; }: Percorre a lista para encontrar o nó com o codigo responsável ao id.
- if(temp == NULL) { return -4; }: Se não encontrar o nó, retorna um sinal de erro.

- if(temp->anterior) { temp->anterior->proximo = temp->proximo; } else { receita->inicio = temp->proximo; } if(temp->proximo) { temp->proximo->anterior = temp->anterior; } else { receita->final = temp->anterior; }: Essa parte do código ajusta os ponteiros anterior e proximo dos nós vizinhos para remover o nó da lista, tratando casos especiais para o primeiro e o último nó.
- while(temp != NULL) { temp->ingrediente.codigo--; temp = temp->proximo; }: Enquanto nosso ponteiro temp não chegar ao final da lista, atualizamos os códigos dos ingredientes restantes na lista.
- free(temp): Libera a memória do nó removido.
- receita->quantidadeIngredientes--: Decremente a quantidadeIngredientes na receita.
- return 0: Retorna 0 para indicar que a remoção deu certo.

int trocar(Receita receita, int cod1, int cod2) tem a função de trocar a posição de dois ingredientes na lista, dados seus códigos.

- if(!receita) { return -1; }: Verifica se a receita existe ou é válida.
- if(receita->quantidadeIngredientes == 0 || receita->quantidadeIngredientes == 1) { return -2; }: Verifica se a lista não está vazia ou se não há apenas um ingrediente na nossa lista, porque se não a função trocar não é executável.
- NoIngrediente no1 = receita->inicio: Ponteiro para o primeiro ingrediente.
- NoIngrediente no2 = receita->inicio: Outro ponteiro para o primeiro ingrediente.
- while(no1->ingrediente.codigo != cod1 || no2->ingrediente.codigo != cod2) {: Enquanto não encontramos ambos os ingredientes.
- if(no1 == NULL || no2 == NULL{ return -4; }: Se percorremos a lista inteira e não encontramos um dos dois, retornamos -4 para indicar erro.
- if(no1->ingrediente.codigo != cod1){ no1 = no1->proximo; }: Se no1 ainda não é o ingrediente com cod1, avançamos no1 para o proximo ingrediente.
- if(no2->ingrediente.codigo != cod2){ no2 = no2->proximo; }: Se no2 ainda não é o ingrediente com co2, avançamos no2 para o proximo ingrediente.
- Ingrediente temp = no1->ingrediente; no1->ingrediente = no2->ingrediente; no2->ingrediente = temp: Troca os dados dos dois ingredientes na lista.
- no1->ingrediente.codigo = cod1; no2->ingrediente.codigo = cod2: Mantém os códigos originais dos ingredientes.
- return 0: Retorna 0 para indicar que a troca deu certo.

int substituir(Receita receita, int cod, Ingrediente ingrediente) tem a função de substituir um ingrediente existente por outro.

- if(!receita) { return -1; }: Verifica se a receita existe ou é válida.
- if(receita->quantidadeIngredientes == 0) { return -2; }: Verifica se a lista não está vazia.
- NoIngrediente atual = receita->inicio: Cria um ponteiro atual para percorrer a lista de ingredientes da receita.
- while(atual->ingrediente.codigo != cod){ atual = atual->proximo; if(!atual){ return -3; }}: Percorre a lista até encontrar o ingrediente com o código especificado.
- atual->ingrediente = ingrediente: Substitui os dados do ingrediente encontrado pelo novo ingrediente.
- atual->ingrediente.codigo = cod: Mantém o código original do ingrediente.

- return 0: Retorna 0 para indicar que a substituição deu certo.

void mostraReceita(Receita receita) tem a função de exibir os detalhes de uma receita, incluindo o nome e a lista de ingredientes.

- if(!receita){ printf("\nErro! Você precisa criar a receita primeiro."); return; }: Verifica se a receita existe ou é válida.
- printf("\n=================================="); printf("\n %s", receita->nomeReceita);
  printf("\n=================================="): Imprime o nome da receita.
- if(receita->quantidadeIngredientes == 0) { printf("\nNão há ingredientes cadastrados."); }: Se não houver ingredientes na receita, imprime uma mensagem informando isso.
- NoIngrediente noAtual = receita->inicio: Cria um ponteiro noAtual para percorrer a lista de ingredientes.
- while(noAtual != NULL){ printf("\n%d - %s - %d %s ", noAtual->ingrediente.codigo, noAtual->ingrediente.nomeIngrediente, noAtual->ingrediente.quantidade, noAtual->ingrediente.medida); if(noAtual->ingrediente.essencial == 1){ printf(""); } noAtual = noAtual->proximo; }: Percorre a lista de ingredientes e imprime os detalhes de cada um. Se o ingrediente for essencial, imprime um asterisco.

void mostraEssenciais(Receita receita) tem a função de exibir apenas os ingredientes marcados como essenciais em uma receita.

- if(!receita){ printf("\nErro! Você precisa criar a receita primeiro."); return; }: Verifica se a receita existe ou é válida.
- printf("\n=================================="); printf("\n %s", receita->nomeReceita);
  printf("\n=================================="): Imprime o nome da receita.
- printf("\n\nIngredientes essenciais:\n"); if(receita->quantidadeIngredientes == 0){ printf("\nNão há ingredientes cadastrados."); }: Se não houver ingredientes na receita, imprime uma mensagem informando isso.
- NoIngrediente noAtual = receita->inicio; int temEssencial = 0: Cria um ponteiro noAtual para percorrer a lista de ingredientes e inicializa uma variável temEssencial para verificar se há ingredientes essenciais.
- while(noAtual != NULL){ if(noAtual->ingrediente.essencial == 1){ printf("\n%d - %s - %d %s ", noAtual->ingrediente.codigo, noAtual->ingrediente.nomeIngrediente, noAtual->ingrediente.quantidade, noAtual->ingrediente.medida); printf("*"); temEssencial = 1; } noAtual = noAtual->proximo; }: Percorre a lista de ingredientes e imprime os detalhes dos ingredientes essenciais. A variável temEssencial é definida como 1 se pelo menos um ingrediente essencial for encontrado.
- if(temEssencial == 0){ printf("Não há ingredientes essenciais nesta receita!"); }: Se não houver ingredientes essenciais, imprime uma mensagem informando isso.

void desalocaReceita(Receita receita) tem a função de desalocar toda a memória alocada para uma receita, incluindo seus ingredientes.

- if (receita == NULL) return: Verifica se a receita é válida (não NULL). Se for NULL, a função retorna imediatamente.
- NoIngrediente atual = receita->inicio: Cria um ponteiro atual para percorrer a lista de ingredientes da receita, começando pelo início da lista.
- NoIngrediente proximo: Declara um ponteiro proximo para armazenar o próximo nó da lista durante a iteração.
- while (atual != NULL) {: Inicia um loop que percorre todos os nós da lista de ingredientes. O loop continua enquanto atual não for NULL.
- proximo = atual->proximo: Armazena o ponteiro para o próximo nó da lista em proximo. Isso é importante porque, uma vez que liberarmos a memória do nó atual, não poderemos mais acessar seu ponteiro proximo.
- free(atual): Libera a memória alocada para o nó atual usando free(). Isso remove o ingrediente da memória.
- atual = proximo: Atualiza o ponteiro atual para o próximo nó (que foi armazenado em proximo). Isso move o loop para o próximo ingrediente na lista.
- free(receita): Após liberar todos os nós da lista de ingredientes, libera a memória alocada para a própria estrutura da receita. Isso remove a receita da memória.

void printCentralizado(const char* texto) tem a função de imprimir um texto centralizado na tela.

- int tamanhoTexto = strlen(texto): Obtém o tamanho do texto usando a função strlen().
- int larguraTela = 80;: Define a largura da tela do terminal como 80 caracteres.
- int espacosEsquerda = (larguraTela - tamanhoTexto) / 2: Calcula o número de espaços necessários à esquerda do texto para centralizá-lo. A divisão por 2 garante que o texto fique centralizado.
- for (int i = 0; i < espacosEsquerda; i++) { printf(" "); }: Imprime os espaços em branco à esquerda do texto. O loop for imprime um espaço para cada posição calculada em espacosEsquerda.
- printf("%s\n", texto);: Imprime o texto usando printf(). O especificador de formato %s indica que uma string deve ser impressa. O caractere \n no final adiciona uma nova linha após o texto, movendo o cursor para a próxima linha do terminal.

5.2 LIVRO

Passando para o arquivo *livro.h* define a struct Livro e os protótipos das funções para manipular o livro de receitas.

struct descritorLivro:

- char nome[50]: Nome do livro de receitas.
- Receita inicio: Ponteiro para a primeira receita do livro.
- int quantidadeReceitas: Número total de receitas no livro.

Protótipo das funções:

- Livro criaLivro(char nome[50]);
- int insereReceita(Livro livro, Receita receita);
- int removeReceita(Livro livro, int posicao);

- Receita recebeReceita(Livro livro, int posicao);
- void mostraLivro(Livro livro);
- void mostraFavoritas(Livro livro);

No arquivo *livro.c* temos a implementação das funções declaradas em *livro.h*:

Livro criaLivro(char nome[50]) tem a função de criar um novo livro de receitas, alocando memória para a struct Livro e inicializando seus campos.

- Livro livro = (struct descritorLivro *)malloc(sizeof(struct descritorLivro)): Aloca memória para a struct Livro.
- strcpy(livro->nome, nome): Copia o nome do livro para o campo nome.
- livro->inicio = NULL: Define o ponteiro para a primeira receita como NULL.
- livro->quantidadeReceitas = 0: Define a quantidade de receitas no livro como 0.
- return livro: Retorna o ponteiro para o novo livro criado.

int insereReceita(Livro livro, Receita receita) tem a função de inserir uma nova receita no livro de receitas

- if(!livro){ return -1; }: Verifica se o livro é válido.
- if(!receita){ return -2; }: Verifica se a receita é válida.
- receita->proxima = livro->inicio: Faz o próximo da nova receita apontar para o antigo início do livro.
- livro->inicio = receita: Atualiza o início do livro para a nova receita.
- livro->quantidadeReceitas++: Incrementa a contagem de receitas no livro.
- return 0: Retorna 0 para indicar que deu certo.

int removeReceita(Livro livro, int posicao) tem a função de remover uma receita do livro de receitas em uma dada posição.

- if(!livro){ return -1; }: Verifica se o livro é válido.
- if(livro->quantidadeReceitas == 0){ return -2; }: Verifica se o livro não está vazio.
- if(posicao < 1 || posicao > livro->quantidadeReceitas){ return -3; }: Verifica se a posição é válida.
- if(posicao == 1){ Receita temp = livro->inicio; livro->inicio = temp->proxima; free(temp); return 0; }: Se a posição for 1, remove a primeira receita do livro.
- Receita atual = livro->inicio; int contador = 1; while((contador + 1) != posicao && atual->proxima != NULL){ atual = atual->proxima; contador++; }: Percorre o livro até a posição anterior à receita a ser removida.
- Receita temp = atual->proxima; atual->proxima = temp->proxima; free(temp); livro->quantidadeReceitas--; return 0: Remove a receita da posição especificada.

Receita recebeReceita(Livro livro, int posicao) tem a função de receber uma receita do livro em uma dada posição.

- if(!livro){ return NULL; }: Verifica se o livro é válido.
- if(livro->quantidadeReceitas == 0){ return NULL; }: Verifica se o livro não está vazio.

- if(posicao < 1 || posicao > livro->quantidadeReceitas){ return NULL; }: Verifica se a posição é válida.
- Receita atual = livro->inicio; int contador = 1; while(contador != posicao && atual != NULL){ atual = atual->proxima; contador++; }: Percorre o livro até a posição desejada.
- return atual: Retorna a receita encontrada na posição especificada.

void mostraLivro(Livro livro) tem a função de exibir os detalhes de um livro de receitas, incluindo o nome do livro e a lista de receitas.

- if(!livro){ printf("\nErro! Você precisa criar o livro de receitas primeiro."); return; }: Verifica se o livro é válido.
- printf("\n=================================="); printf("\n %s", livro->nome); printf("\n=================================="): Imprime o nome do livro.
- printf("\n\nReceitas:\n"); if(livro->quantidadeReceitas == 0){ printf("\nNão há receitas cadastrados."); }: Se o livro estiver vazio, imprime uma mensagem informando isso.
- Receita noAtual = livro->inicio; int contador = 1; while(noAtual != NULL){ printf("\n%d - %s", contador, noAtual->nomeReceita); contador++; if(noAtual->favorita == 1){ printf(" *"); } noAtual = noAtual->proxima; }: Percorre a lista de receitas do livro e imprime o número e o nome de cada receita. Se a receita for favorita, imprime um asterisco.

void mostraFavoritas(Livro livro) tem a função de exibir as receitas favoritas de um livro.

- if(!livro){ printf("\nErro! Você precisa criar o livro de receitas primeiro."); return; }: Verifica se o livro é válido.
- printf("\n=================================="); printf("\n %s", livro->nome); printf("\n=================================="): Imprime o nome do livro.
- printf("\n\nReceitas Favoritas:\n"); if(livro->quantidadeReceitas == 0){ printf("\nNão há receitas cadastrados."); }: Se o livro estiver vazio, imprime uma mensagem informando isso.
- Receita noAtual = livro->inicio; int contador = 1; int temFavorito = 0; while(noAtual != NULL){ if(noAtual->favorita == 1){ temFavorito++; printf("\n%d - %s", contador, noAtual->nomeReceita); printf(" *"); } contador++; noAtual = noAtual->proxima; }: Percorre a lista de receitas do livro e imprime o número e o nome das receitas favoritas.
- if(temFavorito == 0){ printf("\nNão há receitas favoritadas."); }: Se não houver receitas favoritas, imprime uma mensagem informando isso.

5.3 MAIN

O arquivo main.c contém a função que demonstra o uso das funções de manipulação de receitas e livros.

void menu2(Receita receita) exibe o menu de opções para manipulação de uma receita específica.

- int opcao2: Declara uma variável inteira para armazenar a opção escolhida pelo usuário no menu.
- do{ ... } while (opcao2 != 6): Inicia um loop do-while que continua até que o usuário escolha a opção 6 (Sair).
- mostraReceita(receita): Chama a função mostraReceita para exibir os detalhes da receita atual.
- printf(...): Imprime o menu de opções de ingredientes na tela.
- setbuf(stdin, NULL): Limpa o buffer de entrada para evitar problemas com a leitura do scanf.
- scanf("%d", &opcao2): Lê a opção escolhida pelo usuário e armazena em opcao2.
- switch (opcao2): Inicia uma estrutura switch para executar a ação correspondente à opção escolhida pelo usuário.

    1. Case 1: Adiciona um ingrediente à receita.

    - Ingrediente ing: Declara uma variável do tipo Ingrediente para armazenar os dados do novo ingrediente.
    - char nming[50], medida[50]: Declara arrays de caracteres para armazenar o nome e a medida do ingrediente.
    - int qtd, essencial: Declara variáveis inteiras para armazenar a quantidade e a essencialidade do ingrediente.
    - printf(...): Solicita ao usuário que digite o nome, quantidade, medida e essencialidade do ingrediente.
    - setbuf(stdin, NULL); fgets(nming, 50, stdin); nming[strcspn(nming, "\n")] = '\0';: Lê o nome do ingrediente do usuário.
    - scanf("%d", &qtd): Lê a quantidade do ingrediente do usuário.
    - setbuf(stdin, NULL); fgets(medida, 50, stdin); medida[strcspn(medida, "\n")] = '\0';: Lê a medida do ingrediente do usuário.
    - printf(...): Solicita ao usuário que digite se o ingrediente é essencial (0 para não, 1 para sim).
    - scanf("%d", &essencial): Lê a essencialidade do ingrediente do usuário.
    - strcpy(ing.nomeIngrediente, nming); ing.quantidade = qtd; strcpy(ing.medida, medida); ing.essencial = essencial;: Copia os dados do ingrediente para a variável ing.
    - int deuCerto = insereIngrediente(ing, receita): Chama a função insereIngrediente para adicionar o ingrediente à receita.
    - if(deuCerto == -2){...}: Verifica se houve erro de alocação de memória durante a inserção do ingrediente.
    - break: Sai do caso 1.

    2. Case 2: Remove um ingrediente da receita.

    - int cod: Declara uma variável inteira para armazenar o código do ingrediente a ser removido.
    - printf(...): Solicita ao usuário que digite o código do ingrediente que deseja remover.

- scanf("%d", &cod): Lê o código do ingrediente do usuário.
- int deuCerto = removeIngrediente(cod, receita): Chama a função removeIngrediente para remover o ingrediente da receita.
- switch (deuCerto): Inicia uma estrutura switch para lidar com diferentes códigos de retorno da função removeIngrediente.
- case -2: ... break: Imprime uma mensagem de erro se a receita estiver vazia.
- case -3: ... break: Imprime uma mensagem de erro se o código do ingrediente for inválido.
- case -4: ... break: Imprime uma mensagem de erro se o código do ingrediente não for encontrado.
- default: break: Caso padrão, não faz nada.
- break: Sai do caso 2.

3. Case 3: Troca a posição de dois ingredientes na receita.

- int cod1, cod2: Declara duas variáveis inteiras para armazenar os códigos dos ingredientes a serem trocados.
- printf(...): Solicita ao usuário que digite os códigos dos ingredientes que deseja trocar.
- scanf("%d", &cod1); scanf("%d", &cod2): Lê os códigos dos ingredientes do usuário.
- int deuCerto = trocar(receita, cod1, cod2): Chama a função trocar para trocar a posição dos ingredientes.
- switch (deuCerto): Inicia uma estrutura switch para lidar com diferentes códigos de retorno da função trocar.
- case -2: ... break: Imprime mensagem se a receita não tiver ingredientes suficientes.
- case -3: ... break: Imprime mensagem se o valor do código for inválido.
- case -4: ... break: Imprime mensagem se um dos códigos não for encontrado.
- default: break: Caso padrão.

4. Case 4: Exibe os ingredientes essenciais da receita.

- mostraEssenciais(receita): Chama a função mostraEssenciais para exibir os ingredientes essenciais.
- break: Sai do caso 4.

5. Case 5: Substitui um ingrediente existente por outro.

- Declara as variáveis necessárias para o novo ingrediente e solicita as informações ao usuário, depois chama a função substituir
- break: Sai do caso 5.

6. Case 6: Imprime uma mensagem de saída e encerra o loop.

- default: ...: Imprime uma mensagem de opção inválida.

int main() é a função principal do programa.

- Livro livro = cadastrarDados(): Chama a função cadastrarDados() para obter um livro de receitas com dados de teste e armazena o resultado na variável livro.
- int opcao: Declara uma variável inteira para armazenar a opção escolhida pelo usuário no menu principal.
- do { ... } while (opcao != 5): Inicia um loop do-while que continua até que o usuário escolha a opção 5 (Sair).
- mostraLivro(livro): Chama a função mostraLivro para exibir os detalhes do livro de receitas.
- printf(...): Imprime o menu principal do programa na tela.
- setbuf(stdin, NULL);: Limpa o buffer de entrada para evitar problemas com a leitura do scanf.
- scanf("%d", &opcao): Lê a opção escolhida pelo usuário e armazena em opcao.
- switch (opcao): Inicia uma estrutura switch para executar a ação correspondente à opção escolhida pelo usuário.
- return 0: Indica que o programa foi executado com sucesso.

1. Case 1: ... break: Adiciona uma nova receita ao livro.
    - Solicita o nome da receita e se é favorita, chama a função criaReceita e insereReceita e chama a função menu2.
2. Case 2: ... break: Remove uma receita do livro
    - Solicita o número da receita a ser removida, chama removeReceita e exibe o resultado.
3. Case 3: ... break: Exibe uma receita específica do livro.
    - Solicita o número da receita a ser visualizada, chama recebeReceita e chama a função menu2
4. Case 4: ... break: Exibe as receitas favoritas do livro.
    - Chama a função mostraFavoritas.
5. Case 5: ... break: Imprime uma mensagem de saída e encerra o loop.
    - default: ...: Imprime uma mensagem de opção inválida.

5.4 DADOS

O arquivo dados.h define a função cadastrarDados() definida no arquivo dados. c.

- Livro cadastrarDados(): Função para criar um novo livro de receitas com dados predefinidos.
- Ingrediente i,j,k,l,m;: Declaração de variáveis para armazenar informações sobre ingredientes (nome, quantidade, medida, se é essencial).
- Atribuição de valores para os ingredientes i, j, k, l, m (nome, quantidade, medida e se é essencial).

- Criação de três receitas (r, r2, r3) com nomes "Macarrao ao molho branco", "Carbonara" e "Lasanha de frango", marcando a primeira como favorita.
- Inserção dos ingredientes criados nas receitas correspondentes.
- Criação de um livro de receitas chamado "Livro da Paola Caçarola".
- Inserção das três receitas criadas no livro.
- Retorno do livro de receitas preenchido com os dados iniciais.

6. EXEMPLOS DE USO

Exemplo: Criação de um Livro de Receitas e Adição de Receitas.

```c
#include "livroDeReceitas/livro.h"
#include "dados/dados.h"

int main() {
    Livro meuLivro = cadastrarDados(); // Cria e popula o livro com dados de teste
    mostraLivro(meuLivro);             // Exibe o livro com as receitas e ingredientes
    return 0;
}
```

Saída:

```
==================================
      Livro da Paola Caçarola
==================================

Receitas:
1 - Macarrao ao molho branco *
2 - Carbonara
3 - Lasanha de frango

==================================
```

Este exemplo demonstra como o cadastrarDados() cria um livro de receitas chamado "Livro da Paola Caçarola" e o popula com três receitas: "Macarrao ao molho branco" (favorita), "Carbonara" e "Lasanha de frango". Cada receita já contém os ingredientes definidos em dados.c.

O exemplo abaixo mostra como acessar uma receita específica do livro criado por cadastrarDados() e exibir seus ingredientes:

```c
#include "livroDeReceitas/livro.h"
#include "dados/dados.h"

int main() {
    Livro meuLivro = cadastrarDados();
    Receita receita1 = recebeReceita(meuLivro, 1); // Acessa a primeira receita (Macarrao ao molho branco)
    mostraReceita(receita1);                       // Exibe os ingredientes da receita
    return 0;
}
```

Saída:

```
==================================
    Macarrao ao molho branco
==================================

Ingredientes:
1 - paprica - 3 pitadas
2 - noz moscada - 2 nozes *
3 - arroz - 2 xícaras
4 - macarrão - 1 pacote *

==================================
```

Este exemplo demonstra como acessar a primeira receita do livro (Macarrão ao molho branco) e exibir seus ingredientes, incluindo "paprica", "noz moscada", "arroz" e "macarrão". O asterisco indica que "noz moscada" e "macarrão" são ingredientes essenciais nesta receita.

O exemplo a seguir demonstra a remoção de uma receita do livro de receitas e a exibição do livro atualizado:

```c
#include "livroDeReceitas/livro.h"
#include "dados/dados.h"

int main() {
    Livro meuLivro = cadastrarDados(); // Cria e popula o livro com dados de teste

    printf("Removendo a receita 2 do livro...\n");
    removeReceita(meuLivro, 2); // Remove a receita na posição 2 (Carbonara)
    mostraLivro(meuLivro);      // Exibe o livro atualizado

    return 0;
}
```

Saída:

```
Removendo a receita 2 do livro...
==================================
      Livro da Paola Caçarola
==================================

Receitas:
1 - Macarrao ao molho branco *
2 - Lasanha de frango

==================================
```

Neste exemplo, removemos a receita "Carbonara" (que estava na posição 2) do livro. A saída mostra que o livro agora contém apenas "Macarrao ao molho branco" e "Lasanha de frango".

O exemplo abaixo demonstra como trocar e substituir ingredientes em uma receita:

```c
int main() {
    // Cria uma receita de exemplo
    Receita receita = criaReceita("Salada de Frutas", 0);

    // Adiciona alguns ingredientes
    Ingrediente maca = {"Maçã", 2, "unidades", 1};
    Ingrediente banana = {"Banana", 3, "unidades", 1};
    Ingrediente laranja = {"Laranja", 1, "unidade", 1};

    insereIngrediente(maca, receita);    // Código 1
    insereIngrediente(banana, receita);  // Código 2
    insereIngrediente(laranja, receita); // Código 3

    mostraReceita(receita);

    // Troca a posição da Maçã e Laranja
    trocar(receita, 1, 3);
    printf("\nApós trocar Maçã e Laranja:\n");
    mostraReceita(receita);

    // Substitui a Banana por Morango
    Ingrediente morango = {"Morango", 5, "unidades", 0};
    substituir(receita, 2, morango);
    printf("\nApós substituir Banana por Morango:\n");
    mostraReceita(receita);

    return 0;
}
```

Saída:

```
==================================
      Salada de Frutas
==================================

Ingredientes:
1 - Maçã - 2 unidades *
2 - Banana - 3 unidades *
3 - Laranja - 1 unidade *

==================================

Após trocar Maçã e Laranja:
==================================
      Salada de Frutas
==================================

Ingredientes:
1 - Laranja - 1 unidade *
2 - Banana - 3 unidades *
3 - Maçã - 2 unidades *

==================================

Após substituir Banana por Morango:
==================================
      Salada de Frutas
==================================

Ingredientes:
1 - Laranja - 1 unidade *
2 - Morango - 5 unidades
3 - Maçã - 2 unidades *

==================================
```

Neste exemplo, primeiro criamos uma receita "Salada de Frutas" com três ingredientes: Maçã, Banana e Laranja. Em seguida, usamos a função trocar para trocar a posição da Maçã (código 1) com a Laranja (código 3). Finalmente, usamos a função substituir para substituir a Banana (código 2) por Morango. A saída mostra a receita antes e depois de cada operação, demonstrando as alterações na ordem e nos detalhes dos ingredientes.

O exemplo abaixo demonstra a inserção e remoção de receitas do livro de receitas "Livro da Paola Caçarola", criado pela função cadastrarDados().

```c
#include "livroDeReceitas/livro.h"
#include "dados/dados.h"

int main() {
    Livro meuLivro = cadastrarDados(); // Cria e popula o livro com dados de teste

    // Cria uma nova receita
    Receita novaReceita = criaReceita("Salada Caprese", 0);
    Ingrediente tomate = {"Tomate", 2, "unidades", 1};
    Ingrediente mussarela = {"Mussarela de Búfala", 200, "gramas", 1};
    insereIngrediente(tomate, novaReceita);
    insereIngrediente(mussarela, novaReceita);

    // Insere a nova receita no livro
    insereReceita(meuLivro, novaReceita);
    printf("\nLivro após inserir Salada Caprese:\n");
    mostraLivro(meuLivro);

    // Remove a receita "Carbonara" (posição 2)
    removeReceita(meuLivro, 2);
    printf("\nLivro após remover Carbonara:\n");
    mostraLivro(meuLivro);

    return 0;
}
```

Saída:

```
Livro após inserir Salada Caprese:
==================================
      Livro da Paola Caçarola
==================================

Receitas:
1 - Salada Caprese
2 - Macarrao ao molho branco *
3 - Carbonara
4 - Lasanha de frango

==================================

Livro após remover Carbonara:
==================================
      Livro da Paola Caçarola
==================================

Receitas:
1 - Salada Caprese
2 - Macarrao ao molho branco *
3 - Lasanha de frango

==================================
```

Neste exemplo utiliza o livro de receitas "Livro da Paola Caçarola" criado pela função cadastrarDados(). Primeiro, uma nova receita "Salada Caprese" é criada e inserida no livro. Em seguida, a receita "Carbonara" (que está na posição 2) é removida do livro. A saída mostra o estado do livro após cada operação, demonstrando a inserção e remoção de receitas.

Este exemplo demonstra como adicionar e remover ingredientes de uma receita, bem como marcar um ingrediente como essencial, utilizando uma das receitas criadas em dados.c.

```c
#include "receita/receita.h"
#include <string.h>

int main() {
    // Cria uma receita usando os dados de exemplo de dados.c
    Receita receita = criaReceita("Macarrao ao molho branco", 1);

    // Adiciona um ingrediente à receita
    Ingrediente alho = {"Alho", 2, "dentes", 1}; // Marcado como essencial
    insereIngrediente(alho, receita);

    // Exibe a receita com o ingrediente adicionado
    mostraReceita(receita);

    // Remove o ingrediente "arroz" (código 3)
    removeIngrediente(3, receita);
    printf("\nReceita após remover o arroz:\n");
    mostraReceita(receita);

    return 0;
}
```

Saída:

```
==================================
    Macarrao ao molho branco
==================================

Ingredientes:
1 - paprica - 3 pitadas
2 - noz moscada - 2 nozes *
3 - arroz - 2 xícaras
4 - salmão - 350 gramas
5 - macarrão - 1 pacote *
6 - Alho - 2 dentes *

==================================

Receita após remover o arroz:
==================================
    Macarrao ao molho branco
==================================

Ingredientes:
1 - paprica - 3 pitadas
2 - noz moscada - 2 nozes *
3 - salmão - 350 gramas
4 - macarrão - 1 pacote *
5 - Alho - 2 dentes *

==================================
```

Neste exemplo, começamos com a receita "Macarrao ao molho branco" criada com a função cadastrarDados(), que já contém alguns ingredientes. Adicionamos o ingrediente "Alho" como essencial e, em seguida, removemos o ingrediente "arroz" da receita. A saída mostra os ingredientes da receita antes e depois da remoção, demonstrando que o alho foi adicionado corretamente e o arroz removido.

7. CONCLUSÃO

A implementação deste sistema de gerenciamento de receitas e ingredientes apresentou diversos desafios, principalmente no que diz respeito à manipulação de memória e à garantia de que a alocação e desalocação de memória fossem feitas corretamente para evitar vazamentos. A utilização de listas encadeadas duplamente para representar os ingredientes das receitas exigiu um cuidado extra na manipulação dos ponteiros para garantir que a inserção, remoção e troca de ingredientes fossem realizadas de forma eficiente e sem erros.

As principais lições aprendidas incluem:

- A importância de um bom planejamento da estrutura de dados antes de iniciar a implementação do código.
- A necessidade de realizar testes exaustivos para garantir que todas as funcionalidades estejam funcionando corretamente e que não haja vazamentos de memória.
- A importância de escrever um código limpo e bem documentado para facilitar a manutenção e o entendimento do código por outros desenvolvedores.

Em resumo, o desenvolvimento deste projeto proporcionou um aprendizado significativo sobre manipulação de listas encadeadas, alocação dinâmica de memória, desenvolvimento de sistemas de gerenciamento de dados em C e trabalho coletivo.

Gostaríamos de agradecer à professora Maria Adriana por sua orientação e ensinamentos na disciplina de Algoritmos e Estruturas de Dados, que foram essenciais para a realização deste trabalho.