Relatório

O processo de construção do trabalho final de ED1.

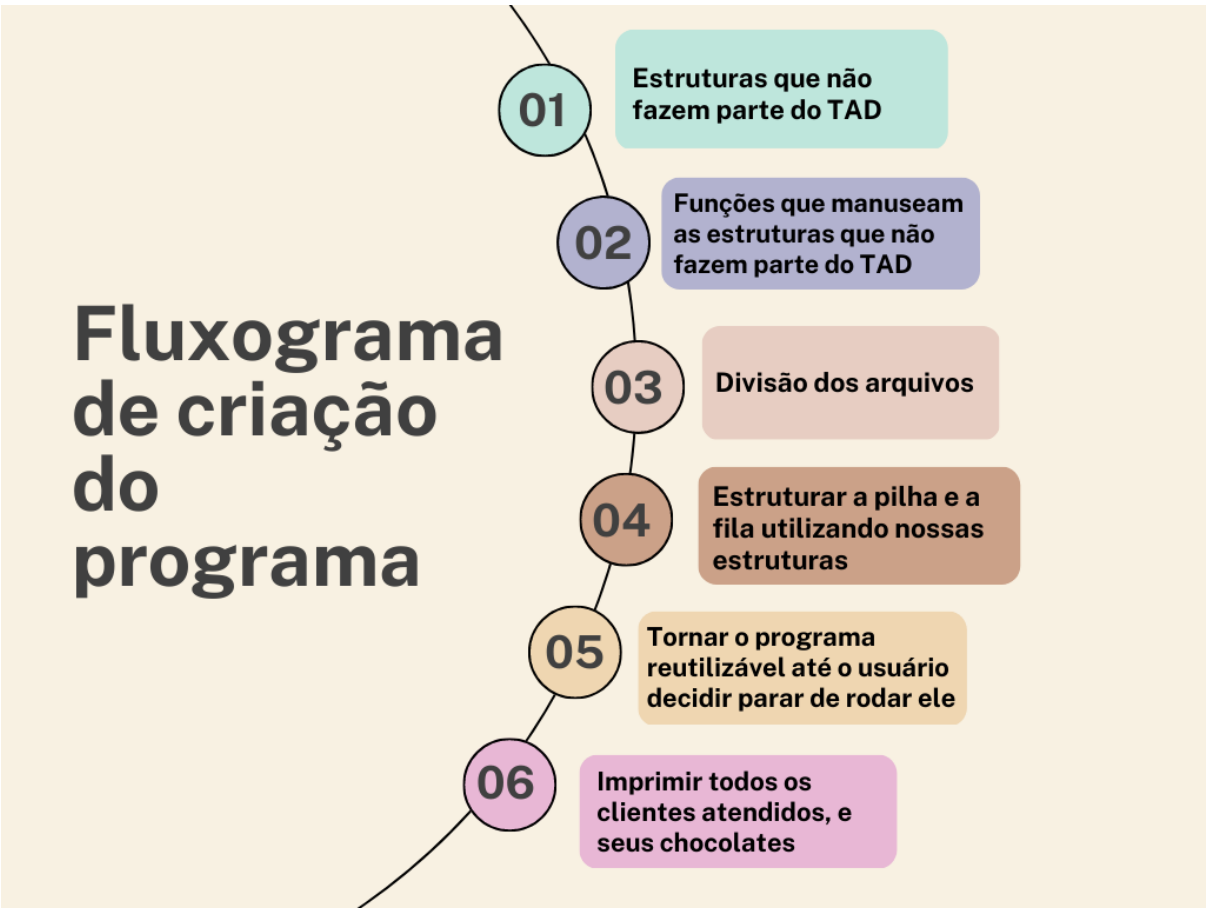
Alunos: Giovanna Maria Alves Evangelista - 12121BSI266
Henrick Oliveira de Souza - 12121BSI237

Professor: Paulo Henrique Ribeiro Gabriel

Uberlândia, 2023.

Introdução

Para a execução do trabalho de construção de um sistema de restaurante usando os conceitos de TAD e estruturas aprendidos em aula, nossa dupla inicialmente criou um roteiro de passos a serem seguidos. Assim, seria possível ter um guia durante toda a construção do projeto,o que facilitaria durante todo o processo criativo e o de implementação prática. Assim, foi elaborado o seguinte fluxograma:



Os passos não foram necessariamente seguidos de forma linear durante a construção do programa como mostrado na imagem acima, no entanto, à medida que decidimos fazer um novo tópico, montamos uma esquematização durante a implementação para que fosse mais didático. Nos tópicos abaixo, abordaremos uma explicação mais detalhada e ampla sobre cada um dos passos listados.

1- Criação das estruturas que não fazem parte do TAD.

Antes de estruturar o TAD, nos achamos mais conveniente fazer as estruturas que o programa utilizaria, uma vez que seria a base para diversos outros passos do sistema. Dessa forma, foram criadas as estruturas Cliente, Item e Menu. Para isso, criamos um arquivo .h que funcionou como biblioteca no programa principal, abrigando todas as estruturas citadas anteriormente.

A estrutura menuItem registra um dos itens que é possível escolher dentro do menu, o que tornou-a mais complexa de pensar, uma vez que ela seria compartilhada entre as outras duas. Em seu interior, ela abriga um int, correspondente ao ID do produto do cardápio, uma string, contendo o nome desse item e um float com o preço.

```
struct menuItem {
    int itemId;
    char productName[20];
    float price;
```

Já a Menu, abriga as opções de escolha que o cliente possui. Inicialmente, foi pensado que poderíamos fazer duas coisas separadas para executar a função que precisaríamos, sendo uma para colocar o ID do item e outra para o vetor de itens, contudo, comprovamos através de testes que não seria eficiente. Assim, optamos por fazer a estrutura Menu em si, que tem o vetor de itens e dois *int*, sendo um para a posição de pôr o novo item e outro para o ID. Ambos os *int* acrescentam 1 a cada item que é adicionado.

```
struct menu{
    MenuItem fullMenu[15];
    int posToAddNewItem;
    int newItemId;
```

A Cliente foi criada com o intuito de ser usada dentro do TAD, para que fosse possível fazer a fila de clientes que seriam atendidos de acordo com a sua ordem de chegada. Na sua execução, colocamos o ID do cliente (o número de sua comanda) e a quantidade de pedidos por meio de *int*, junto uma string do chocolate que cada cliente recebeu e os produtos que foram pedidos do tipo menuItem.

```
struct client {

    int clientIndex;
    char recivedChocolate[20];
    int quantityOfOrders;
    MenuItem orders[15]; //15 é
```

2- Criação das funções que manuseiam as estruturas sem ser do TAD.

Para fins de testes, para saber como utilizar e se as estruturas funcionavam da forma que precisávamos, começamos a fazer as funções que seriam mais simples de serem elaboradas, como as de imprimir, que basicamente utilizam apenas printf (como exemplo ordersOf, ou a printMenu). Depois de várias tentativas e de entender como utilizar cada uma das estruturas, nós começamos a dividir o programa e a iniciar as partes que executariam os principais comandos ditados no enunciado.

3- Divisão dos arquivos.

Depois de aprender como importar arquivos em C, achamos conveniente dividir as estruturas pessoais, nesse processo, a gente já gerou os arquivos .c e .h, tanto da pilha, quanto da fila, e das estruturas, pra ficar mais fácil de manusear e achar o que precisava de conserto durante os testes e execução do trabalho. Dessa maneira, ao final, obtivemos o total de seis arquivos, sendo eles:

- queue.c
- queue.h
- stack.c
- stack.h
- structs.h
- Trabalho Final.C, que contém o conteúdo principal do programa.

4- Criação e estruturação da pilha e da fila utilizando as estruturas feitas no passo 1.

Com as estruturas e as funções que as manuseiam, nossa dupla começou com a construção do TAD pilha, e em seguida, com o TAD fila. Optamos por seguir nessa ordem porque tínhamos em mente que se algo desse errado, seria em alguma das duas estruturas de dados citadas anteriormente, o que possibilitaria checar facilmente e fazer as correções ou mudanças necessárias somente nos arquivos convenientes.

4.1 - A Pilha.

A pilha foi um pouco trabalhosa de ser implementada no cenário do restaurante, uma vez que ela seria basicamente, uma pilha de strings. Como a linguagem C não possui um tipo string, foi preciso utilizar uma matriz de char, onde cada posição “i” guardava uma string que representava um chocolate.

Utilizamos a estrutura de pilha básica que o professor apresentou em sala de aula para criarmos a nossa, adaptando para a necessidade que teríamos conforme pedido no enunciado do trabalho em questão. Portanto, a nossa pilha ao final foi constituída pelas funções criarPilha, destruirPilha, esvaziarPilha, verificação de pilha cheia, empilhar e desempilhar.

Nesse processo, observamos um padrão que todas as funções do string.h demandam um ponteiro de char, e, para fazer elas, é necessário declarar um vetor de char, exemplo[20]. Para atribuí-lo a string, se passa somente “exemplo”. Logo, chegamos a conclusão que, quando se declara um vetor, a variável acompanhada das posições é um ponteiro para uma sequência de bits.

4.2- A Fila.

Na criação do TAD fila, também seguimos a estrutura dada pelo professor em sala de aula, adaptando para o contexto que precisaríamos utilizar. Nessa estrutura de dados, colocamos as funções: criarFila, destruirFila, esvaziarFila, enfileirar e desenfileirar. Ela serviu como organização do sistema de atendimento das pessoas que chegariam no restaurante, cujo o primeiro a chegar seria o primeiro a ser atendido, e assim por diante.

Implementar a fila sem dúvidas foi a coisa mais complexa de ser feita durante a criação do trabalho, dado que ela guardava diversos dados das pessoas que foram atendidas. Dentro da fila, há um vetor de estruturas cliente, que possui em seu interior um vetor de estruturas item. Esse sistema fez com que caíssemos várias vezes em um erro nomeado como *segmentation fault,* até que fosse possível achar a forma correta de se enfileirar e retirar cada cliente da fila durante a execução do programa.

4.3 - Utilização dos TAD na Main.

Depois de fazer todos os ajustes e testes para ver se as estruturas de dados Pilha e Fila funcionavam em todos os casos necessários para o bom funcionamento do programa, passamos a dar um pouco de foco para a execução da parte Main no arquivo principal. Para seguir as instruções de que nenhuma das funções que manuseiam a Pilha e a Fila podiam ter printf, foram feitas funções na main que são capazes de mostrar os integrantes de ambos os TAD como solução para atender o requisito.

Obs: como um adendo “charmoso” a pilha se imprime na ordem na qual os chocolates serão retirados (último a entrar, primeiro a sair).

5 - Tornar o programa reutilizável até o usuário decidir parar de rodar ele.

Para ser condizente com algo que seria verdadeiramente usado no dia-dia e que pudesse ser usado de forma dinâmica e sem que se encerrasse assim que o usuário terminasse de executar uma função, optamos por fazer um switch case que tomaria conta de todas as operações na interface e serviria como um menu de escolhas.

Dessa forma, cada número corresponde a uma função que a pessoa que está utilizando o programa pode efetuar. Confira abaixo:

- 0: Encerrar o dia
- 1: Imprimir o Menu
- 2: Adicionar item ao Menu
- 3: Imprimir a Fila
- 4: Imprimir a Pilha
- 5: Adicionar chocolate na Pilha
- 6: Criar novo cliente
- 7: Atender novo cliente
- 8: Limpar tela

Tornou-se necessário tratar casos de três das funções acima que possibilitaria erros no programa se não tivessem recebido a devida atenção. São eles:

5.0 - Caso 0 ao 8.

Como mostrado acima, o menu de escolhas possui apenas 9 (nove) opções que podem ser escolhidas pelo usuário durante o programa. Portanto, se em um cenário hipotético ele decidisse colocar um número maior ou igual a 9, haveria um erro na execução por essa escolha não estar disponível. Assim, optamos por fazer uma condicional para que se esse caso ocorresse, o programa dispararia um aviso e impediria que o usuário inserisse uma opção inválida, voltando ao menu principal para que ele repetisse a ação.

5.1 - Caso 5

O caso 5 trata da adição de chocolates na pilha. Utilizando lógica básica, se a pilha está cheia, não adicione mais chocolates, então por meio de um operador ternário, construímos uma trava que checa que, se o booleano worked é 0(false) ele para o caso, e não quebra a pilha. Se for um cenário contrário em que não seja o caso da Pilha estar cheia, ele é adicionado. Ambos os casos tem uma mensagem para informar o usuário da ocorrência.

5.2 - Caso 6

De todas as funções do menu de escolhas, “criar novo cliente”, foi a mais difícil de implementar, por ser uma sequência extensa até adicionar o cliente na fila.

Primeiramente checamos se a fila está cheia, e caso a resposta for positiva, o comando não funciona, e dispara um aviso pedindo para atender um cliente da fila primeiro, que só assim irá liberar uma posição para inserir um novo. Caso não esteja cheia, aumenta-se em 1 o índex do cliente, e realoca uma só estrutura dedicada a criar clientes, a newClient.

Ademais, para adicionar um pedido ao novo cliente da fila, vem uma função um pouco complexa, que é a de adicionar pedido ao cliente, cuja pega o menu do dia, faz o uso da função de imprimir ele, e, de acordo com o índex do item que o cliente escolher, adiciona o item todo ao vetor de pedidos do cliente. Logo após, vem um campo para saber se o cliente deseja outro pedido, e se afirmativa a resposta, essa função é chamada novamente. Os casos de parada dela são, quando o cliente faz 15 pedidos, que é o limite que ele pode fazer, ou quando é inserido o char “N” que significa nesse caso, “não”. Dessa forma, quando essa repetição encerra, pegamos todos os dados desse cliente e o enfileiramos.

5.3 - Caso 7:

O caso 7 trata de atender o cliente, ou seja, tirá-lo da fila. Como todos os clientes precisam receber chocolates, antes de iniciar o caso, é checado se a pilha está vazia, e se ela estiver, é necessário que seja inserido um chocolate nela, para dar ao cliente.

Além disso, é imprescindível ter alguém na fila, que é checado logo após a execução. Caso essas duas condições sejam cumpridas, a gente atribui o cliente a uma variável local, chamada dequeuedClient, e copia no campo de chocolate recebido dela, o que sair da pilha que contém todos os doces. Logo, adicionamos em um vetor e em uma variável que serão melhor tratados no próximo passo.

6- Imprimir todos os clientes atendidos e seus chocolates

Um dilema que adquirimos ao final se dava em imprimir todos os atendimentos do dia. Para que isso fosse possível, as pessoas teriam que estar armazenadas em algum lugar, no entanto, o vetor convencional só deixa armazenar uma quantidade previamente determinada, o que era inviável já que não é possível determinar a demanda. Inicialmente pensamos em implementar uma lista encadeada, contudo, encontramos adversidades e não conseguimos fazê-la funcionar perfeitamente bem no programa.

Assim, para conseguir imprimir o vetor de Clientes por inteiro, que foi declarado com 50 (cinquenta) posições, declaramos outra variável, para saber em qual das posições será colocado para outro cliente, que é a clientsOfTheDay. Ela consiste em um inteiro que conta os clientes do dia e é adicionada após o caso 7, desse modo, ao decidir fechar pelo dia, antes de finalizar, ocorre uma repetição que imprime todos os clientes do dia, que já terão obrigatoriamente um chocolate vinculado a eles.