Trabalho 1 EDA1
Autor (individual): Isaac Menezes Pereira, 190088885.

--> Romantissismo:
Nos longos idos do trabalho de fim de semestre da primeira disciplina de programação para computadores, o desafio era
outro: criar um sistema em C para distribuição das marmitas na UnB. Requisitos: cadastrar cozinheiro, cadastrar
marmitas, cadastrar aluno, registrar pedido, excluir cozinheiro e excluir aluno. No fim, gerar um arquivo com todas
as mudanças.

Agora, com a nostalgia dos velhos tempos, surge uma nova ideia: fazer um sistema que distribui uma
pilha de marmitas prontas para uma fila de até 200 alunos famintos. Caso sobrem marmitas, há prejuízo para o RU.
Caso faltem marmitas, os alunos famintos processam (com razão) o RU. No fim do dia, haverá lucro ou o restaurante irá
à falência? Só você, cozinheiro, poderá "decidir".

--> Instruções de compilação e execução:
Para compilar: gcc -Wall t1.c pilhaStringsHeader.c filaStringsHeader.c -o t1.x
Para rodar: ./t1.x
Observação: os arquivos pilhaStringsHeader.c, pilhaStringsHeader.h, filaStringsHeader.c e filaStringsHeader.x devem
estar no mesmo diretório que t1.c, caso contrário, o caminhos desses arquivos devem ser adicionados no comando de
compilação.

--> Melhores explicações de funcionamento:
O arquivo t1.c funciona assim:
- A função main() cria uma stack para as marmitas (flavorStack) e uma queue para as matrículas dos alunos (porque
assim serão representados) (idQueue);
- Após isso, chama menu();
- Em menu() há uma verificação para checar se uma stack de marmitas (flavorStack) já foi criada (na função produce())
e se essa stack já foi "distriuída" (na função distribute()), ou seja, se as marmitas já foram distribuídas. Caso já
haja uma stack de marmitas (flavorStack), sua opção é distribuí-la ou encerrar o programa. Caso você já a tenha
distribuído, não é possível recriá-la ou redistribuí-la. Essa verifcação acontece checando se a stack de marmitas
(flavorStack) já foi criada ou com o retorno da função distribute(). Só é possível recriar uma stack de marmitas
(flavorStack) e redistribuí-la caso o programa seja executado novamente.
- A função produce() cria uma matriz de strings e depois a adiciona na stack de marmitas (flavorStack).É necessário
definir os sabores das marmitas e a quantidade total para produção. A função longString() verifica se o tamanho do
nome do sabor da marmita é menor que 100. Já a função removeSpaces() remove eventuais espaços em branco do nome do

sabor das marmitas;
- A função distribute() cria um arquivo final de log (logMealBoxes.txt), que também é verificado com a função
verifyFile(). Após isso, a função escolhe uma quantidade aleatória de alunos que comparecerão ao restaurante, bem como
uma quantidade aleatória para um valor padrão para processo de todos os alunos. Essas escolhas são feitas com a
variável jocker (inicializada com 200). Depois disso as matrículas dos alunos (id's) são geradas com a função
generateId() e, então, inseridas na queue de alunos (idQueue). Após isso, há uma grande verificação para decidir: se
guess é maior que quantStudents, se guess é menor que quantStudents ou se são iguais. A variável guess guarda um chute
do usuário para a quantidade de alunos que comparecerão ao restaurante. A variável quantStudents é definida por
jocker. A variável guess define o tamanho de flavorStack, já a variável quantStudents define o tamanho de idQueue.
Isto é, se guess > quantStudents, a stack de sabores (flavorStack) é maior que a queue de estudantes (idQueue). Se
isso, então um loop acontece levando em consideração os índices de flavorStack e são retornados os elementos de
flavorStack e de idQueue ao mesmo tempo enquanto um index é menor que a quantidade de elementos de idQueue. Caso esse
index for maior que a quantidade de elementos de idQueue, então são retornados apenas os elementos de flavorStack com
o seguinte dizer: "prejuizo". Ações semelhantes são feitas caso guess < quantStudents e se guess == quantStudents e
são necessárias para evitar que sejam feitas requesições dos elementos de uma das duas listas (flavorStack ou idQueue)
onde já não hajam índices;
- No final, há o arquivo logMealBoxes.txt como reposta final informando seu lucro, prejuízos por ser processado e
prejuízo caso haja disperdício de marmitas. Na verdade, é uma grande piada nostalgica.
- Observação: até aqui, onde há a palavra "verificação" é esperado que o programa resista a uma eventual entrada
inesperada. Ou seja, se já há uma stack de marmitas (flavorStack), novamente, suas opções são distribuí-la ou encerrar
o programa. No entanto, se, em algum momento o usuário digitar um char ou um char[] onde um valor esperado é um int, o
programa não necessariamente persistirá. Motivo: na verdade, não deu tempo de implementar esse tipo de proteção;
- Para facilitar, há um diagrama chamado "árvore de código" que descreve graficamente a ordem de chamadas das funções,
bem como um printScreen de um exemplo de execução do programa feito pelo autor.

--> O que é esperado?
Uma dica de fluxo seguro onde é garantido o funcionamento correto:
- Executar a main();
- Ao ver o primeiro output de menu(), escolher a primeira opção;
- Definir o total de sabores das marmitas (um número inteiro);
- Definir cada sabor para as marmitas;

- Definir o valor de cada marmita, um valor obrigatoriamente padrão (um número de ponto flutuante);
- Definir o total da produção (um número inteiro);
- Ao ver novamente o output de menu(), escolher a segunda opção;
- Ao ver o output da segunda opção, bem como o output de menu(), escolher a terceira opção que encerrará o programa;
- Checar o arquivo logMealBoxes.txt.

Observação: há 1 (um) 200 (duzentos) avos de chance de obter lucro máximo. Caso haja lucro máximo: printf("\a");