Um final em Terabithia

Todo mundo quer aprender a trabalhar com código, mas nós vamos além do código. Sua capacidade de analisar um problema é a habilidade mais importante que você pode desenvolver. Os problemas abaixo possuem um contexto mais rebuscado, mas vamos encontrar soluções.

Hoje vamos desenvolver um programa feito com vários outros programas utilizando um conceito de software modulado, feito com partes independentes.

O objetivo é desenvolver uma plataforma para funcionários do hotel usarem, não os hóspedes.

Introdução

Antes de tudo: execute o código passado proposto e veja como ele se comporta.

Depois disso... De acordo com cada uma das solicitações abaixo, adapte o programa principal:

1. ~~Invente um nome para o Hotel.~~
2. ~~Ao acessar o sistema, **exiba** "Bem-vindo ao {Hotel}".~~
3. ~~Ao acessar o sistema, **pergunte o nome do usuário e uma senha**. O nome do usuário não precisa de validação. A senha deve ser **2678**.~~
4. ~~Na função "inicio", **utilize escolha/caso** (switch/case) para validar a opção escolhida pelo do usuário.~~
5. ~~**Sempre que o usuário acessar o sistema, diga** "Bem-vindo ao Hotel {Hotel}, {Nome}. É um imenso prazer ter você por aqui!".~~
6. ~~**Substituir a expressão {Hotel}** pelo nome do hotel informado em todos os pontos do código.~~
7. ~~Sempre que o usuário sair do sistema, **exiba a mensagem** "Muito obrigado e até logo, {Nome}."~~
8. Para cada escolha que o usuário fizer no menu principal, **deve ser desenvolvido um programa seguindo as instruções abaixo**. Esse programa deve ser criado utilizando uma função principal como chamada. Todo o restante é feito com sua liberdade.
9. Considere que o hotel possui 20 quartos e ao iniciar o programa **todos estão livres.**
10. **Atualize o menu de opções e a função inicio** com todas as opções de programas abaixo**.**
11. **Atualize a função "erro"** com as novas opções do menu**.**
12. Ao encerrar qualquer programa abaixo, **sempre retorne ao menu inicial.**

Subprograma 1

Desenvolva um programa que:

1) Receba o valor de uma diária no hotel e a quantidade de dias de hospedagem. Valide as informações, ou seja, impeça que o usuário informe dados inválidos, de maneira que o valor da diária não seja negativo e que a quantidade de dias não seja nem negativa, nem maior que 30.

Em caso de informação inválida escreva na tela “Valor Inválido” e volte ao início do programa.

2) Em seguida, caso o usuário tenha informado um valor correto, pergunte o nome do hóspede.

3) Agora será informado o número do quarto (de 1 a 20); A informação deve ser armazenada e se outro hóspede tentar ocupar um quarto já ocupado o sistema informará “Quarto já está ocupado”. No caso de um quarto ocupado, deve ser oferecido ao usuário a escolha de outro quarto.

4) O próximo passo é perguntar se o usuário confirma a reserva. Caso não aceite, volte ao menu inicial.

Detalhes técnicos:

Camada A — Reforço de validação e estados

1. **Validação forte de entrada**
    - Diária deve ser numérica e maior ou igual a 0. Também deve aceitar virgula ou ponto como separadores e **armazenar em centavos** (evitar float).
    - Dias: inteiro em [1..30]. Zero, negativos, >30 ou não numéricos → “**Valor inválido, {Nome}**” e retorna ao início do **programa 1** (não derruba o sistema).
    - Número do quarto: inteiro em [1..20]. Fora da faixa → “**Quarto inexistente**”.
    - Confirmação só aceita S/N (case-insensitive). Qualquer outra coisa re-pergunta.
2. **Estados de quarto**
    - Estados possíveis: livre, ocupado, pré-reserva.

- Ao escolher um quarto **ainda não confirmado**, marque como pré-reserva; se o usuário responder "N", volte o estado para livre.
- Ao confirmar (S), muda para ocupado.

3. **Sugestão inteligente de quarto**
    - Se o quarto escolhido estiver ocupado, exiba **até 3 alternativas** mais próximas numericamente (por distância absoluta) ainda livres. Ex.: “O 7 está ocupado. Alternativas: 6, 8, 9.”
4. **Listagem final**
    - Após confirmar (ou desistir), sempre liste o mapa: 1-livre; 2-ocupado; ... exatamente como no enunciado.

Critérios de aceitação (A)

- Não existem floats em cálculo final (apenas centavos).
- pré-reserva nunca fica “presa” após desistência.
- Sugestões aparecem em ordem de proximidade.

Camada B — Regras de preço e políticas

1. **Política de preço**
    - **Desconto por estadia longa**:
        - 5–9 dias: −8%
        - 10–30 dias: −15%
    - **Taxa municipal**: R$ 7,50 **por dia** (fixa, sem desconto).
    - **Depósito de garantia**: 30% do **total com taxa** (arredonde para centavos, regra bancária).
    - Exibir memória de cálculo:
      Subtotal = diária*dias → desconto → + taxa municipal → Total → Depósito (30%).
2. **Regras de nome**
    - Nome do hóspede: mínimo 3 caracteres úteis (trim), proibir só espaços e números puros.
3. **Rollback de erros**

- **Qualquer** erro depois de marcar pré-reserva (ex.: falha na confirmação, cancelamento ou cálculo) **libera** o quarto automaticamente antes de voltar ao menu.

Critérios de aceitação (B)

- O total bate com a memória de cálculo em todos os casos de teste.
- Depósito sempre é 30% do total final (após desconto + taxa), arredondado corretamente.

Camada C — Robustez e UX de sistema

1. **Segurança no login**
    - Senha 2678, até **3 tentativas** por acesso. Ao exceder: volta ao menu "login" e registra "tentativas excedidas". (Não encerra o app.)
2. **Auditoria mínima**
    - Log interno (em memória) por ação: {timestamp, usuario, acao, detalhes}.
    - Ações: LOGIN_OK/FAIL, PRE_RESERVA, CONFIRMA_RESERVA, CANCELA_RESERVA, ERRO_VALIDACAO.
3. **Persistência de sessão**

- O mapa de 20 quartos vive enquanto o sistema está rodando (não reinicia ao voltar ao menu).
- Implementar função pura resumirOcupacao(quartos[]) -> string usada pelo menu e pelos módulos (facilita teste).

Critérios de aceitação (C)

- Após 3 falhas de senha, novo login é exigido e o estado dos quartos permanece intacto.

Subprograma 2 - Cadastro de hóspedes (Modelo 1).

Imagine que uma família acaba de chegar ao balcão do hotel e quer se hospedar.

Primeiro diremos ao programa o valor padrão da diária e só então cadastraremos todos os hóspedes.

O programa receberá vários nomes de hóspedes e suas idades, sem limites.

Caso o hóspede tenha menos de 6 anos, ela ou ele não paga a diária – nesses casos mostre na tela "[Nome do hóspede] possui gratuidade".

Caso o hóspede tenha mais de 60, ela ou ele paga metade da diária – mostre na tela "[Nome do hóspede] paga meia".
O usuário informará hóspedes até digitar a palavra "PARE", que interrompe a entrada de dados.

Ao fim, mostre a quantidade de gratuidades, a quantidade de meias hospedagens e o valor total, considerando todos os hóspedes informados.

Detalhes técnicos:

Camada A — Validação e Controle de Fluxo

1. **Validação forte**
    - Valor da diária: numérico positivo, aceitando vírgula ou ponto, armazenado em **centavos** (evitar float).
    - Nome do hóspede: mínimo 3 caracteres úteis (sem só espaços, nem apenas números).
    - Idade: inteiro ≥ 0 e ≤ 130 (limite humano razoável). Idades inválidas → mensagem de erro e repergunta sem perder os já cadastrados.
2. **Mensagens personalizadas**
    - Gratuidade: [Nome] possui gratuidade (idade [idade]).
    - Meia diária: [Nome] paga meia (idade [idade]).
    - Pagamento integral: [Nome] paga diária integral (idade [idade]).
3. **Encerramento seguro**
    - Palavra-chave PARE só é aceita se for digitada **sozinha** (case-insensitive).
    - Se for digitada como nome de hóspede → ignora (pode ter um hóspede chamado "Pare").

Camada B — Políticas e Relatórios Detalhados

1. **Política de cobrança**
    - Gratuidade: 0% da diária.
    - Meia diária: 50% da diária.
    - Integral: 100% da diária.
    - Aplicar **taxa municipal fixa** de R$ 7,50/dia por hóspede (gratuidade isenta da taxa).
2. **Resumo detalhado**
   Ao final, mostrar:
    - Lista com: Nome, Idade, Tipo de Cobrança, Valor final por hóspede (BRL).
    - Quantidade de gratuidades, meias e integrais.
    - **Total geral** (já com taxas).
    - Valor médio pago por hóspede **pagante** (desconsiderando gratuidades).
3. **Desconto de grupo**
    - Se o total de hóspedes pagantes ≥ 5 → aplicar desconto de 10% sobre o total final. Mostrar o desconto e o total com desconto.

Camada C — Robustez, Persistência e Auditoria

1. **Persistência temporária**
    - Lista de hóspedes armazenada em memória para consultas posteriores **enquanto o sistema estiver rodando**.
    - Possibilidade de acessar um **relatório anterior** sem precisar recadastrar.
2. **Auditoria**
    - Logar cada cadastro como {timestamp, usuario, nomeHospede, idade, tipoCobranca, valorCalculado}.
3. **Funcionalidade extra**
    - Adicionar **busca por hóspede**: usuário digita um nome parcial e o sistema retorna todos os hóspedes com esse padrão no nome, junto com o valor pago.

- Ignorar diferenças de maiúsculas/minúsculas na busca.

Casos de Teste Rápidos

1. **Entrada inválida**: diária -10 → “Valor inválido, {Nome}” e re-pergunta.
2. **Idade fora do intervalo**: idade -1 ou 200 → erro e re-pergunta.
3. **Palavra “pare” no nome**: Parecida Maria não encerra cadastro.
4. **Grupo grande**: 6 hóspedes pagantes → desconto de grupo aplicado.
5. **Busca**: digitar mar retorna “Maria, Marcelo, Omar”.

Subprograma 3 - Cadastro de hóspedes (Modelo 2)

Como no programa anterior, imagine que uma família acaba de chegar ao balcão do hotel e quer se hospedar.

Monte um programa em que o usuário poderá cadastrar e pesquisar hóspedes.

O programa deve oferecer um menu com algumas opções ao usuário: *1- cadastrar; 2- pesquisar; 3-listar; 4- sair*.

A opção “cadastrar” deve permitir que o usuário informe um nome de hóspede, gravando-o em memória (máximo de 15 cadastros; caso atinja essa quantidade, mostre “Máximo de cadastros atingido”).

A opção “pesquisar” deve permitir ao usuário informe um nome e, caso seja encontrado um nome **exatamente igual**, mostre a mensagem “Hospede (nome) foi encontrado". Se o nome não foi encontrado mostre “Hóspede não encontrado”.

A opção “listar” exibe todos os hóspedes do hotel um a um.

O programa deve permitir que o usuário realize essas operações repetidas várias vezes, até que use a opção “4”, que retorna ao menu principal.

Detalhes Técnicos

Camada A — Validação e Fluxo

1. **Menu dinâmico**
    - Mostrar sempre o número de cadastros já ocupados e vagas restantes.
    - Se já houver 15 hóspedes cadastrados, a opção **“cadastrar”** fica desabilitada (mostrando “Indisponível”).
2. **Validação de nomes**
    - Nome mínimo de 3 caracteres úteis (trim), proibido duplicar exatamente o mesmo nome (ignorar maiúsculas/minúsculas na comparação).
    - Mensagem especial para duplicados: "{Nome} já está cadastrado."
3. **Pesquisa exata**
    - Continua exigindo **igualdade exata** (case-insensitive), mas avisa também a posição na lista caso encontrado:
      "Hóspede João encontrado na posição 4."

4. **Listagem formatada**
    - Mostra os hóspedes numerados (1 - João, 2 - Ana, ...).

Camada B — Funcionalidades Extras

1. **Pesquisa parcial (extra)**
    - Além da pesquisa exata, oferecer sub-opção para pesquisa **por parte do nome** (contém).
    - Ex.: procurar "mar" retorna “Maria”, “Marcos”, “Omar”.
2. **Remover hóspede**
    - Nova opção no menu: 5 - remover.
    - Ao remover, recompactar a lista (sem “buracos”) e informar quantas vagas estão livres agora.
    - Caso o nome não exista: "Hóspede não encontrado para remoção."
3. **Ordenação opcional**
    - Sub-opção dentro de “listar”:
        - 1 - ordem de cadastro (original)
        - 2 - ordem alfabética (A-Z, ignorando acentos e maiúsculas/minúsculas).

Camada C — Robustez, Auditoria e Persistência

1. **Persistência temporária**
    - A lista de hóspedes vive enquanto o sistema está rodando (não reinicia ao voltar ao menu principal).
2. **Auditoria de operações**
    - Log interno: {timestamp, usuario, acao, detalhe} para cada operação:
        - CADASTRO_OK, CADASTRO_DUPLICADO, PESQUISA_OK, PESQUISA_FALHA, REMOCAO_OK, REMOCAO_FALHA, LISTAGEM.
3. **Relatório final antes de sair**
    - Ao escolher a opção “4 - sair”, mostrar:
        - Total de hóspedes cadastrados.

- Quantos foram adicionados na sessão.
- Quantos foram removidos na sessão.
- Quantas pesquisas foram realizadas.

Casos de Teste Rápidos

1. **Lista cheia**: cadastrar até 15 → tentar cadastrar o 16º → “Máximo de cadastros atingido” e opção “cadastrar” desabilitada.
2. **Duplicado**: “Ana” cadastrado → tentar “ana” → “Ana já está cadastrado.”
3. **Pesquisa exata**: procurar “Carlos” → encontrado na posição correta.
4. **Pesquisa parcial**: procurar “mar” → retorna “Maria, Marcos, Omar”.
5. **Remoção**: remover “Pedro” e listar → não aparece mais, vagas aumentam.
6. **Ordenação**: listar em A-Z ignora acentos (Érica → depois de Daniel).

Subprograma 4

Neste cenário, o hotel receberá não hóspedes, mas eventos.

Eventos são grandes e complexos, por isso precisamos pensar em diversas situações.

Importante: as quatro partes abaixo constituem um único programa.

Parte 1: Onde será o evento?

O auditório Laranja conta com **150** lugares e espaço para até **70** cadeiras adicionais.

O auditório Colorado conta com **350** lugares, sem espaço para mais cadeiras.

Desenvolva um programa que **receba o número de convidados** do evento e faça uma verificação sobre a quantidade: se for maior que **350** ou se for menor que zero, mostre a mensagem ***"Número de convidados inválido"***. Se o valor informado é válido, mostre na tela qual dos auditórios é o mais adequado, dando prioridade ao Auditório Laranja.

No caso do auditório Laranja, ainda, calcule quantas cadeiras adicionais serão necessárias, observando o limite citado acima.

Exemplo 1:

Programa pergunta => "Qual o número de convidados para o seu evento?"

Resposta do usuário => 360

Programa exibe => "Quantidade de convidados superior à capacidade máxima".

Exemplo 2:

Programa pergunta => "Qual o número de convidados para o seu evento?"

Resposta do usuário => 192

Programa exibe => "Use o auditório Laranja (inclua mais 42 cadeiras)"

Programa exibe => "Agora vamos ver a agenda do evento."

Exemplo 3:

Resposta do usuário => "Qual o número de convidados para o seu evento?"

Resposta do usuário => 300

Programa exibe => "Use o auditório Colorado"

Programa exibe => "Agora vamos ver a agenda do evento."

Parte 2: Quando será o evento?

O hotel oferece reserva de seu auditório caso o contratante necessite. O auditório está disponível para reservas de segunda a sexta das ***7hs às 23hs***; sábados e domingos apenas das ***7hs às 15hs***.

1. Monte um programa que receba o dia da semana em texto.

Importante: na entrada de dados para dia da semana, desconsidere acentos e use letra minúscula. Não é necessário validação para isso no código.

2. Também receba a hora (número inteiro, desprezando minutos e segundos)
3. Diga se o auditório está disponível ou não de acordo com as regras especificadas acima.
4. Quando o auditório estiver disponível, receba ainda o nome da empresa e mostre na tela a mensagem *"Auditório reservado para (nome da empresa): (dia da semana) às (horas)hs"*.

Exemplo 1:

Programa pergunta => "Qual o dia do seu evento?"

Resposta do usuário => sabado

Programa pergunta => "Qual a hora do seu evento?"

Resposta do usuário => 16

Programa exibe => Auditório indisponível"

Exemplo 2:

Programa pergunta => "Qual o dia do evento?"

Resposta do usuário => segunda

Programa pergunta => "Qual é a hora do evento?"

Resposta do usuário => 13

Programa pergunta => "Qual o nome da empresa?"

Resposta do usuário => Lojas Transilvânia

Programa exibe => "Auditório reservado para Lojas Transilvânia. Segunda às 13hs."

Parte 3: Quantos trabalharão no evento?

Quando o hotel se prepara para eventos, são oferecidos garçons para servir os convidados. Considerando que cada garçom custa **R$ 10,50** por hora, escreva um programa que receba o número de garçons necessários e o total de horas do evento. Para cada **12** convidados é necessário um garçom, sempre arredondando para cima, e para cada duas horas de evento adicione mais um garçom.

Depois calcule o custo total que o hotel terá com a contratação desses profissionais e exiba o resultado em tela.

Exemplo:

Programa pergunta => "Qual a duração do evento em horas?"

Resposta do usuário => 8

Programa exibe => "São necessários 48 garçons."

Programa exibe => "Custo: R$ 504.0"

Programa exibe => "Agora vamos calcular o custo do buffet do hotel para o evento."

Parte 4: E quanto ao Buffet?

O hotel oferece café, água e salgados para cada um dos convidados de um evento alocado em seus salões. A quantidade de café, em litros, é calculada como 0,2 litro para cada convidado; a quantidade de água é calculada como 0,5 litro para cada convidado; são oferecidos 7 salgados por pessoa.

Com a quantidade de convidados informada na Parte 1, calcule a quantidade de água, café e salgados para o evento, mostrando em tela esses valores.

Sabendo que cada litro de café custa, **0,80 centavos**, cada litro de água custa **0,40 centavos** e o cento de salgados custa **34 reais**, calcule o custo total com comida do evento.

Exemplo:

Programa exibe => "O evento precisará de 20 litros de café, 50 litros de água, 700 salgados."

Agora, exiba o relatório com custos e pergunte se o usuário aceita todos os valores informados.

Exemplo:

Evento no Auditório Colorado.

Nome da Empresa: Hotel Transilvânia.

Data: Segunda, 13H às 21h.

Duração do evento: 8H.

Quantidade de garçons: 48.

Quantidade de Convidados: 192

Custo dos garçons: R$504.00

Custo do Buffet: R$274.00

Valor total do Evento: R$ 778.00

Programa pergunta => "Gostaria de efetuar a reserva? S/N"

Resposta do usuário => S

Programa exibe => "{Nome}, reserva efetuada com sucesso."

Se "SIM", exiba a mensagem *"Reserva efetuada com sucesso"*.

Caso contrário, exiba a mensagem *"Reserva não efetuada."*

Detalhes Técnicos

Camada A — Validação e Encadeamento de Módulos

1. **Validação reforçada**
    - Parte 1: Convidados devem ser inteiros entre 1 e 350. Zero e negativos → inválidos.
    - Parte 2: Dia deve ser convertido para minúsculo e sem acento **automaticamente**.
    - Parte 3: Horas de duração devem ser inteiros positivos (mínimo 1).

- Parte 4: Custos calculados em **centavos** para evitar problemas com ponto flutuante.

2. **Encadeamento condicional**
    - Se a Parte 1 reprovar (capacidade inválida), não prosseguir.
    - Se a Parte 2 reprovar (indisponibilidade), voltar para a escolha de data/hora sem repetir a Parte 1.
    - Cada parte deve poder ser chamada separadamente em testes.
3. **Prioridade de uso do Laranja**
    - Mesmo se o Colorado comportar, usar o Laranja sempre que possível sem ultrapassar o limite de cadeiras extras (70).
    - Mensagem clara: "Use o auditório Laranja (inclua mais X cadeiras)" ou "Use o auditório Colorado".

Camada B — Regras Extras e Relatórios

4. **Regras adicionais para garçons**
    - Cálculo inicial: arredondar(convidados / 12).
    - Regra extra: para **cada bloco de 2 horas completas** no evento, adicionar 1 garçom extra **sobre o total já calculado**.
    - Permitir cálculo manual para comparação, mas mostrar qual é o cálculo oficial do sistema.
5. **Buffet com regras adicionais**
    - Café: 0,2 L por convidado, Água: 0,5 L, Salgados: 7 por convidado.
    - Arredondar líquidos para **uma casa decimal**, salgados para o inteiro mais próximo.
    - Cálculo de custo:
        - Café: R$ 0,80/L
        - Água: R$ 0,40/L
        - Salgados: R$ 34,00 por 100 unidades
    - Mostrar **memória de cálculo**:
    - Café: 38,4 L × R$0,80 = R$30,72

- Água: 96 L × R$0,40 = R$38,40
- Salgados: 1344 unidades × R$0,34 = R$457,00

6. **Relatório detalhado**
    - Mostrar no final todos os dados, incluindo:
        - Auditório escolhido e cadeiras extras
        - Nome da empresa
        - Dia e hora do evento
        - Duração
        - Número de garçons
        - Quantidade de convidados
        - Custos de garçons, buffet e total

Camada C — Funcionalidades Avançadas e Auditoria

7. **Sugestão automática de horário**
    - Se o dia/hora informado não estiver disponível, sugerir **até 3 alternativas** mais próximas (mesmo dia e horário válido mais próximo, ou dia alternativo com o mesmo horário).
8. **Histórico/Auditoria**
    - Cada tentativa de reserva (mesmo que rejeitada) é armazenada com {timestamp, usuario, sucesso/falha, motivo, custo_total}.
    - Possibilidade de listar reservas anteriores na mesma execução.
9. **Confirmação com tempo limitado**
    - Após exibir o custo total, o usuário tem **30 segundos** para responder S ou N (simulação com timeout).
    - Se não responder no tempo, a reserva é cancelada automaticamente com a mensagem "Tempo esgotado. Reserva não efetuada."
10. **Múltiplas reservas na mesma execução**
    - Sistema volta ao menu inicial após cada reserva (aceita ou rejeitada) até que o usuário opte por sair.

Casos de Teste Rápidos

1. **Número de convidados inválido**: 360 → “Quantidade de convidados superior à capacidade máxima”.
2. **Uso do Laranja com cadeiras extras**: 192 → Laranja + 42 cadeiras.
3. **Colorado direto**: 300 → Colorado.
4. **Dia/hora fora do horário**: sábado, 16h → indisponível, sugerir sábado 15h ou domingo 15h.
5. **Cálculo garçons**: 192 convidados, 8h duração → (192/12)=16 + (8/2)=4 extras = 20 garçons.
6. **Buffet arredondado**: valores corretos e arredondados conforme regras.
7. **Confirmação fora do tempo**: não responde → reserva cancelada.

Subprograma 5

O hotel tem carros para levar seus hóspedes a passeios. O carro é sempre abastecido pelo hotel que tem convênios com dois postos de Gasolina: o **Wayne Oil** e o **Stark Petrol**. Os carros podem ser abastecidos tanto com álcool quanto gasolina, mas os preços têm flutuado bastante, então é necessário que que um funcionário cheque qual o posto mais em conta para abastecimento.

Para isso, é necessário desenvolver um programa em que o usuário informe os valores de álcool e gasolina dos dois postos e depois calcule qual o combustível mais atraente e o posto mais barato. Considere que o tanque do carro comporta 42 litros de combustível e esse **sempre** será o volume a ser abastecido. Considere que quando o álcool estiver 30% mais barato que a gasolina, é mais barato abastecer com álcool.

Detalhes Técnicos:

Camada A — Comparação e Cálculo Rigoroso

1. **Entrada com validação**
    - Preço não pode ser negativo nem zero.
    - Valores devem ter no máximo 2 casas decimais (senão, arredondar).
2. **Cálculo correto da vantagem do álcool**
    - Regra: álcool é mais vantajoso se preco_alcool <= preco_gasolina × 0.70.
    - Deve ser calculada separadamente para cada posto.
3. **Cálculo do custo total para abastecer 42 L**
    - Para cada posto e cada combustível, calcular:

      `custo = preco_combustivel × 42`
4. **Identificar melhor opção**
    - Primeiro, decidir combustível mais vantajoso em cada posto.
    - Depois, comparar custos finais entre os postos para indicar o mais barato geral.

Camada B — Regras e Relatórios Extras

1. **Empate de custo**
    - Se dois postos derem o mesmo custo final (até 2 casas decimais), mostrar mensagem:
      "Tanto faz, o custo é o mesmo nos dois postos."
      e listar ambos.
2. **Relatório detalhado**
    - Mostrar tabela final com:

Posto	Combustível	Preço/L	Custo Total
Wayne Oil	Álcool	4.20	R$ 176,40
Wayne Oil	Gasolina	5.82	R$ 244,44
Stark Petrol	Álcool	4.35	R$ 182,70
Stark Petrol	Gasolina	6.17	R$ 259,14

    - Mostrar também: “**No Wayne Oil**, vale mais a pena abastecer com álcool/gasolina.” para cada posto

Casos de Teste:

1. **Álcool mais barato no Wayne Oil**: gasolina 5.00, álcool 3.40 → álcool vence (3.40 ≤ 5.00 × 0.70).
2. **Gasolina mais barata no Stark**: Wayne com preços altos, Stark com gasolina menor preço.
3. **Empate perfeito**: todos preços iguais → tanto faz.
4. **Simulação de variação**: álcool sobe 10%, gasolina cai 5% → recalcular.

Exemplo:

```
Programa pergunta => "Qual o valor do álcool no posto Wayne Oil?"
Resposta do usuário => 4.20
Programa pergunta => "Qual o valor da gasolina no posto Wayne Oil?"
Resposta do usuário => 5.82
Programa pergunta => "Qual o valor do álcool no posto Stark Petrol?"
Resposta do usuário => 4.35
Programa pergunta => "Qual o valor da gasolina no posto Stark Petrol?"
```

Resposta do usuário => 6.17
Programa exibe => "{Nome}, é mais barato abastecer com gasolina no posto Wayne Oil

Dados para comparação:

Posto	**Combustível**	**Preço/L**	**Custo Total**
Wayne Oil	Álcool	4.20	R$ 176,40
Wayne Oil	Gasolina	5.82	R$ 244,44
Stark Petrol	Álcool	4.35	R$ 182,70
Stark Petrol	Gasolina	6.17	R$ 259,14

".

Subprograma 6

Um hotel contrata empresas terceirizadas para realizar a manutenção de seus aparelhos de ar-condicionado.
Algumas empresas oferecem desconto quando o serviço é realizado em uma quantidade mínima de aparelhos. Além disso, certos prestadores acrescentam uma **taxa de deslocamento** fixa caso o valor final (já com desconto) seja inferior a um limite mínimo.

Dentro desse contexto, crie um programa em que:

1. Para cada empresa, solicite ao usuário:
   - Nome da empresa.
   - Valor do serviço por aparelho.
   - Quantidade de aparelhos a serem mantidos.
   - Percentual de desconto (que pode ser zero).
   - Quantidade mínima de aparelhos para que o desconto seja aplicado.
   - Valor mínimo para dispensar a taxa de deslocamento.
   - Valor da taxa de deslocamento (que será somada apenas se o valor final ficar abaixo do mínimo).

2. Calcule
   - Valor bruto = valor por aparelho × quantidade de aparelhos.
   - Aplique o desconto apenas se a quantidade de aparelhos for maior ou igual à quantidade mínima para desconto.
   - Após o desconto, verifique se o valor final é menor que o mínimo exigido para não pagar deslocamento; se for, adicione a taxa de deslocamento.

3. Mostre para cada empresa:

```
O serviço de [nome da empresa] custará R$ [valor final calculado]
```

4. Permita que o usuário informe dados de várias empresas, repetindo o processo até que responda 'N' à pergunta:

```
Deseja informar novos dados? (S/N)
```

5. Ao final:
   - Liste todos os orçamentos informados, do menor para o maior valor.
   - Mostre a mensagem:

```
O orçamento de menor valor é o de [nome da empresa] por R$ [menor valor]
```

Observações:

- Sempre serão informadas **pelo menos duas empresas**.
- O programa deve validar para que nenhum valor monetário seja negativo e que as quantidades sejam números inteiros positivos.
- O cálculo deve ser feito em função/procedimento, recebendo os dados e devolvendo o valor final.
- Não utilize bibliotecas prontas para ordenação; implemente a ordenação manualmente.

Exemplo:

Programa pergunta => "Qual o nome da empresa?"
Resposta do usuário => Empresa 1
Programa pergunta => "Qual o valor por aparelho?"
Resposta do usuário => 100
Programa pergunta => "Qual a quantidade de aparelhos?"
Resposta do usuário => 7
Programa pergunta => "Qual a porcentagem de desconto?"
Resposta do usuário => 12
Programa pergunta => "Qual o número mínimo de aparelhos para conseguir o desconto?"
Resposta do usuário => 3
Programa exibe => "O serviço de Empresa 1 custará R$ 1350.0"

Programa pergunta => "Deseja informar novos dados, {Nome}? (S/N)"
Resposta do usuário => S

Programa pergunta => "Qual o nome da empresa?"
Resposta do usuário => Empresa 2
Programa pergunta => "Qual o valor por aparelho?"
Resposta do usuário => 95
Programa pergunta => "Qual a quantidade de aparelhos?"
Resposta do usuário => 6
Programa pergunta => "Qual a porcentagem de desconto?"
Resposta do usuário => 9
Programa pergunta => "Qual o número mínimo de aparelhos para conseguir o desconto?"
Resposta do usuário => 10

Programa exibe => "O serviço de Empresa 2 custará R$1480.0"

Programa pergunta => "Deseja informar novos dados, {Nome}? (S/N)"

Resposta do usuário => N
Programa exibe => "O orçamento de menor valor é o [Empresa 1 por R$ 1200.0]"