



UNIDADE CURRICULAR: Introdução à Programação

CÓDIGO: 21173

DOCENTE: José Coelho

A preencher pelo estudante

NOME: Daniel Junior

N.º DE ESTUDANTE: 2304335

CURSO: LEI – Licenciatura Engenharia Informática

Atenção: apenas é permitida a utilização de funções standard do C, que constem em Anexo p-fólio: Funções standard mais utilizadas

Autoavaliação:

Critérios	Alínea A	Alínea B	Alínea C	Alínea D
Funcionalidade				
Qualidade				
Relatório dimensão / forma				
Relatório conteúdo				
Relatório testes				

Autoavaliação de acordo com os critérios de correção: +0.1 na nota do e-fólio
Ver critérios de correção no espaço da UC ([aqui](#))

TRABALHO / RESOLUÇÃO:

Contexto e Dificuldades

Este trabalho envolveu a implementação incremental do jogo "Sequência", através da implementação paulatina (*sempre quis usar esta palavra*) das alíneas A, B, C e D, cada uma adicionando novas funcionalidades. Infelizmente, houve pouca clareza nos requisitos, levando à necessidade de acompanhar o fórum em busca de esclarecimentos. A repetição de dúvidas em diferentes posts gerou atrasos e incertezas, e não chegou ao meu conhecimento qualquer compensação de tempo oriundo do enunciado e seu esclarecimento. Apesar disso, as quatro alíneas foram concluídas, mantendo o código simples e claro, sem recorrer a recursos complexos de C ou funções não permitidas. O foco foi cumprir o mínimo necessário, de forma incremental, garantindo que cada alínea utilizasse ou estendesse as soluções anteriores.

Abaixo seguem meus comentários na estrutura recomendada, aplicando-a a cada alínea. Apresento apenas a informação nova que cada alínea trouxe em relação às anteriores. Ao final, forneço um anexo com testes, incluindo simulações completas de output do terminal, especialmente para as alíneas C e D, conforme solicitado.

Alínea A – estado: terminado

Estrutura de dados:

- `int k;`
 - Guarda o valor K, entre 2 e 100.
 - É lido no início do programa e utilizado para calcular a sequência inicial.

Abstrações funcionais:

- Função para ler K, validar intervalo e imprimir sequência inicial de 2 números.
 - Exemplo: `lerValorDeK()`, `validarIntervaloK(k)` e `mostrarSequenciaInicial(k)`
 - Faz apenas impressão direta, sem lógica complexa.

Funcionamento global:

- `main`: lê K, valida, imprime sequência inicial composta por dois números ($K/2$ e $K-(K/2)$). Encerra em seguida.

Alternativas ponderadas:

- Poderíamos ter escolhido outra forma de imprimir a sequência inicial (por exemplo, sempre imprimir "K/2" duas vezes se K for par), mas isso não traria vantagem.
- Decidimos manter a divisão simples ($K/2$ e $K-(K/2)$).

Notas:

- Sem problemas significativos nesta fase, apenas a insegurança sobre casos limite como $K=2$ ou $K=100$.
- Sem consultas ao fórum, pois a tarefa era simples.

Alínea B – estado: terminado (informação adicional à A)

Estrutura de dados (novidade da B):

- `int sequencia[100];`
 - Armazena a sequência lida do utilizador.
 - Domínio: números inteiros positivos.

Abstrações funcionais (novidade da B):

- Funções para calcular soma, produto, soma das diferenças absolutas.
- Função para validar a sequência (`calcularSoma(sequencia, tamanhoSequencia)`, `calcularProduto(sequencia, tamanhoSequencia)`).
- Função para verificar vitória ($difabs = K$, onde $difabs$ é a soma das diferenças absolutas).

Funcionamento global (alínea B):

- `main`: além de ler K, lê a sequência, valida-a, e imprime "Sequencia invalida", "Sequencia vitoria" ou "Sequencia valida".

Alternativas ponderadas (alínea B):

- Poderíamos ter usado estruturas mais complexas (listas, structs especializados), mas um array simples é suficiente.
- Não consideramos outras formas de verificação devido ao limite de tempo e foco na simplicidade.

Notas (alínea B):

- Falta de clareza quanto a sequências vazias ou valores extremos. Decidi ignorar casos não especificados, para esta alínea.
- Fórum não esclareceu totalmente detalhes como comportamento exato se a sequência tiver apenas um número, mas segui a interpretação direta.

Alínea C – estado: terminado (informação adicional à B)

Estrutura de dados (novidade da C):

- Reutilizamos o array `sequencia[100]`.
- Adicionamos variáveis globais:
 - `int JOGADOR_ATUAL;` (0 ou 1, onde 0 é o jogador A, e 1 é jogador B)
 - `int CONTAGEM_JOGADAS;`
 - `int INSERIU_OU_REMOVEU;`

Abstrações funcionais (novidade da C):

- Função `lerJogada(sequencia, &tamanho)`: lê índice e valor, e aplica inserção, remoção ou substituição conforme regras.
 - Se índice \geq tamanho, adiciona no final.
 - Valor 0 remove, valor negativo insere valor absoluto, valor positivo substitui.
 - Inserir/remover mantém a vez apenas uma vez; segunda inserção/remoção passa a vez.
 - Substituir ou adicionar no final passa a vez imediatamente.
- Função `processarTurno()`: verifica se deve passar a vez ou não.

Funcionamento global (alínea C):

- `main`: agora entra em loop de jogadas, validando a cada jogada e verificando se há vitória, derrota (sequência inválida) ou empate após K jogadas.

Alternativas ponderadas (alínea C):

- Poderia usar estruturas dinâmicas para inserção/remoção mais fácil, mas manter arrays e deslocar elementos é simples e claro.
- Poderia ter tentado interpretações diferentes para índices e valores, mas optamos pela mais lógica.
- Poderia também ter utilizado uma máquina de estados para cada um dos estados, e aplicado *design patterns*, tais como *strategy*, *sequence* e outras... mas tornar-se-ia por demais complicado e muito complexo para apenas um ficheiro, optei por seguir o princípio *KISS*.

Notas (alínea C):

- Falta de clareza gerou tempo extra analisando o fórum. Mesmo assim, sem esclarecimentos oficiais (no próprio enunciado do efolio), coletei e assumi convenções simples:
 - Índice negativo = 0
 - Índice igual ou maior que tamanho = adicionar no final
 - Valor 0 remove
 - Valor <0 insere em dado índice
 - Valor >0 substitui
- Esta alínea obrigou-me a uma grande reescrita e repensar na abordagem do problema, para tentar melhor escolher o momento de processar um turno. A impressão da sequência inicial antes da primeira jogada, como algo independente. A pesquisa de *design patterns*, e o abandona devido à complexidade de implementação em C.

Alínea D – estado: terminado (informação adicional à C)

Estrutura de dados (novidade da D):

- Mesmo array e variáveis globais.
- Agora, se jogada = (-2 -2), ativamos o jogador artificial, que testa jogadas hipotéticas.

Abstrações funcionais (novidade da D):

- Função `jogadaArtificial(...)`: tenta remover, reduzir, aumentar ou inserir de forma heurística, procurando vitória primeiro ou primeira jogada válida que resulte em menor sequência.
- Função `testarMovimento(...)`: aplica jogada em cópia da sequência para verificar se é válida ou vitoriosa sem alterar estado real.

Funcionamento global (alínea D):

- `main`: igual à C, mas se a jogada for (-2 -2), chamamos `jogadaArtificial` para decidir o movimento.
- Caso a IA encontre jogada vitoriosa ou válida, aplica. Caso contrário, pode não fazer nada.

Alternativas ponderadas (alínea D):

- Poderia implementar uma heurística mais sofisticada, mas isso aumentaria complexidade.
- Considerar menos tentativas de jogadas para acelerar decisão. Escolhi iterar até encontrar algo ou esgotar opções.

Notas (alínea D):

- Determinar limites da IA, complexidade envolvida, se deveria recorrer à APIs externas. Acho que acabei por *overcomplicate* o que deveria ser um simple conjunto de testes.
- A heurística não é ideal, mas cumpre o mínimo exigido.
- Tentei manter o mais próximo possível com a Alínea C. Não compreendi muito bem a idéia de fazer reuso das funções neste efolio, uma vez que pediram 4 ficheiros em separados, um para cada alínea.
 - Acabei por fazer *copy-paste* na mesma, ferindo *DRY*.

Caso	Entrada	Saída	Resultado
01	-1	K tem de ser entre 2 e 100.	Alínea A: Test: $k < 0$ Output: Erro
02	0	K tem de ser entre 2 e 100.	Alínea A: T: $k == 0$ O: Erro
03	10	Sequencia: 5 5	Alínea A: T: $2 \leq k \leq 100$ O: 5 5
04	101	K tem de ser entre 2 e 100.	Alínea A: T: $k > 100$ O: Erro
05	10 5 5 0	Sequencia valida	Alínea B: T: sequencia valida O: sequencia valida
06	10 6 6 0	Sequencia invalida	Alínea B: T: sequencia invalida, pela soma. O: sequencia invalida
07	10 1 1 8 0	Sequencia invalida	Alínea B: T: sequencia invalida pelo produto. O: sequencia invalida
08	10 5 4 1 0	Sequencia valida	Alínea B: T: sequencia valida com foco na soma das dif. Absolutas. O: sequencia valida
09	10 0 3 1 2	Indique K: 10 Sequencia: 5 5 [Joga A] Jogada (indice valor): Sequencia: 3 5 [Joga B] Jogada (indice valor): Sequencia: 3 2 Jogador B perdeu.	Alínea C: T: Jogador B perde com sequencia invalida. O: Jogador B perde.
10	10 3 2	Indique K: 10 Sequencia: 5 5 [Joga A] Jogada (indice valor): Sequencia: 5 5 2 Jogador A perdeu.	Alínea C: T: Jogador A entra sequencia invalida e perde imediatamente O: Jogador A perde.
11	10 1 5 1 5 ... 1 5	Indique K: 10 Sequencia: 5 5 [Joga A] Jogada (indice valor): Sequencia: 5 5 [Joga B] Jogada (indice valor): ... Sequencia: 5 5 [Joga B] Empate.	Alínea C: T: Jogadas repetidas ate um empate.
12	10 -2 -2 0 0	Indique K: 10 Sequencia: 5 5 [Joga A] Jogada (indice valor): Jogada artificial: 0 2 Sequencia: 2 5 [Joga B] Jogada (indice valor): Sequencia: 5 Jogador B perdeu.	Alínea D: T: AI realiza jogada valida. O: Jogada valida de A, e B perde por jogada invalida.
13	10 -2 -2 1 6 -2 -2	Indique K: 10 Sequencia: 5 5 [Joga A] Jogada (indice valor): -2 -2 Jogada artificial: 0 2 Sequencia: 2 5 [Joga B] Jogada (indice valor): 1 6 Sequencia: 2 6 [Joga A] Jogada (indice valor): -2 -2 Jogada artificial: 1 -1 Sequencia: 2 1 6 Jogador A ganhou.	Alínea D: T: AI pode ganhar um jogo sozinha. O: AI ganhou