E-fólio B | Instruções para a realização do E-fólio

LABORATÓRIO DE PROGRAMAÇÃO | 21178

Período de realização

15 de maio a 24 de maio de 2026

Data limite de entrega

24 de maio de 2026, até às 23:59 de Portugal Continental

Temática

Legibilidade, qualidade e reutilização de código & Testes sistemáticos

Objetivos

- Melhorar a legibilidade e documentação do código (Módulo 3);
- Implementar testes de unidade e integração (Módulo 4);
- Garantir a qualidade e robustez do programa desenvolvido;
- Refletir criticamente sobre o processo de melhoria e os resultados dos testes.

Cenário: GreenTrack - Evolução do Sistema

No E-fólio A desenvolveu uma versão corrigida e modularizada do sistema GreenTrack. Agora pretende-se evoluir esse sistema, aplicando boas práticas de legibilidade, documentação e testes sistemáticos.

O código que entregou no E-fólio A é a base de partida obrigatória para este trabalho. Se recebeu *feedback* do avaliador sobre o E-fólio A, pode (e deve) incorporar essas correções no código de partida. No relatório, assinale que melhorias decorrem desse *feedback* e quais são as novas no âmbito da Tarefa 1, mostrando o estado inicial (a sua entrega original) e o estado após a intervenção.

Deverá:

1. Melhorar a legibilidade do código (nomes, comentários, estrutura);
2. Documentar as *interfaces* de forma completa;
3. Implementar testes de unidade e de integração para validar o funcionamento;
4. Analisar e refletir criticamente sobre o processo de melhoria e sobre o que os testes revelaram.

Num contexto em que ferramentas de IA assistem à escrita de código, a capacidade de avaliar legibilidade, detectar problemas e validar comportamento com testes é o que distingue o engenheiro de um utilizador passivo dessas ferramentas.

Nota: Pode reorganizar a arquitetura do projeto entregue no E-fólio A, caso considere necessário para melhorar a legibilidade, modularidade ou testabilidade.

Tarefas a desenvolver

Para uma execução eficiente, é fundamental que a Tarefa 1 (Melhoria da legibilidade e documentação) seja concluída e estabilizada antes de iniciar a Tarefa 2 (Implementação de testes). A refatoração do código, tais como a alteração de nomes de funções ou de variáveis para nomes mais expressivos, após a escrita dos testes poderá invalidar os testes, obrigando a trabalho duplicado de correção.

Tarefa 1: Melhorias de legibilidade e documentação (1,2 valores)

Para fundamentar as suas melhorias, reveja os recursos formativos do Módulo 3, em especial o RF 3.3 | Legibilidade de Código em Linguagem C, o RF 3.1 | Considerações para desenvolvimento das atividades formativas e RF 3.2 | Modularidade em Programas de Média Dimensão.

a) Reveja todo o código do E-fólio A e aplique melhorias de legibilidade:
   - Nomes de variáveis, funções e módulos expressivos e consistentes;
   - Formatação padronizada;
   - Simplificação de funções complexas (princípio da responsabilidade única);
   - Uso de *guard clauses* para reduzir aninhamento;
   - Etc…

b) Documente todas as *interfaces* públicas (ficheiros .h) e eventualmente os ficheiros .c:
   - Propósito da função;
   - Parâmetros (o que são, pré-condições);
   - Retorno (o que significa, possíveis erros);
   - Efeitos colaterais (se aplicável).

c) Adicione comentários que expliquem as decisões de implementação, não apenas o que o código faz.

d) No relatório, apresente:
   - As principais alterações realizadas;
   - Exemplos concretos de código antes/depois. O estado inicial apresentado na tabela deve corresponder ao código efetivamente entregue no E-fólio A. O avaliador verificará essa correspondência;
   - Justificação das decisões tomadas e alternativas que considerou.

   Pode usar uma tabela, por exemplo, com este formato:

Localização (Módulo/Função)	Estado inicial (E-fólio A)	Melhoria efetuada (E-fólio B)	Justificação técnica e alternativas
módulo X / função A	if (a) { if (b) { ... } }	Guard clause: if (!a) return; if (!b) return; …	Reduz o nível de aninhamento, tornando o fluxo principal imediatamente visível. Alternativa considerada: manter a estrutura original com comentários - rejeitada porque não elimina a complexidade visual.
módulo Y / função B	int x = 0; x++;	int total_registos = 0; total_registos++;	Nome expressivo elimina a necessidade de comentário explicativo. Alternativa: adicionar comentário junto à variável - rejeitada porque comenta o óbvio em vez de tornar o código auto-documentado.

Nota: se o seu código do E-fólio A já estava legível e documentado, indique explicitamente como o garantiu durante o desenvolvimento. Essa reflexão é avaliada.

Tarefa 2: Testes de unidade (1,0 valor)

a) Crie um conjunto de testes de unidade para as funções críticas do sistema:
   - Pelo menos 5 funções testadas, distribuídas pelos diferentes módulos;
   - Para cada função, cubra: caso normal, caso limite e caso de erro;
   - Caso os testes revelem falhas, descreva o processo de diagnóstico e correção. Caso não tenham revelado falhas funcionais, reflita criticamente sobre como os testes aumentaram a confiança no comportamento do sistema e que limitações ainda subsistem;
   - Um *printscreen* do terminal que mostre o teste a falhar antes da correção. A ausência desta demonstração de diagnóstico resultará em penalização. Se nenhum teste de unidade revelou falha, inclua o *printscreen* do *run* completo com todos os resultados OK e justifique por que considera a cobertura suficiente.

Pode usar uma tabela com este formato:

Função	Tipo de caso	Caso de Teste	Porquê este valor? (Justificação)	*Printscreen*	Resultado esperado	Resultado obtido
função_A	normal	valor típico válido	Verificar o comportamento correto no cenário de utilização comum	-	sucesso (1)	sucesso (1)
função_B	limite	MAX - 1	Testar o comportamento imediatamente antes de atingir a capacidade máxima do sistema	-	sucesso (1)	sucesso (1)
função_C	erro	valor inválido (ex: -1)	Verificar que a função rejeita corretamente input fora do domínio esperado	ver imagem X do relatório ou inserir aqui	erro (0)	erro (0)

b) Escolha um dos seus testes de unidade. Se alterasse, por exemplo, o valor de entrada por N+1 e N-1, o comportamento da função mudaria? Explique com base no seu código concreto.

c) Utilize assert() combinado com printf para diagnóstico, conforme o padrão do RF 4.3:

```
int obtido   = funcao_a_testar(entrada);
int esperado = 42;
if (obtido != esperado)
    printf("FALHA: funcao_a_testar(%d) esperado %d, obtido %d\n",
           entrada, esperado, obtido);
assert(obtido == esperado);
```

d) Organize os testes em ficheiros separados por módulo (ex.: test_planta.c, test_rega.c);

e) Para cada teste, indique no relatório:
   - Objetivo do teste;
   - Relevância (o que valida e porque é importante);
   - Resultado inicial (antes de eventuais correções ao código);
   - Se o teste revelou algum problema e como foi corrigido.

Nota: Não é esperada uma *framework* de testes completa nem cobertura exaustiva do sistema. Os testes devem demonstrar compreensão dos princípios de validação e não quantidade excessiva de casos.

Estrutura recomendada para cada ficheiro de testes:

```
// test_planta.c  -  exemplo de estrutura de teste
#include <assert.h>
#include <stdio.h>
#include "planta.h"

void test_adicionar_planta_valida() {
    /* Arrange */
    resetar_plantas();// Função auxiliar que cada estudante deve criar/adaptar
para limpar o estado antes do teste

    /* Act */
    int resultado = adicionar_planta(1, "Rosa", "Rosa canina",
                                     "10/04/2026", 3);

    /* Assert */
    int obtido = total_plantas();
    int esperado = 1;
    if (obtido != esperado)
        printf("FALHA: test_adicionar_planta_valida - "
               "esperado %d, obtido %d\n", esperado, obtido);
    assert(obtido == esperado);
    printf("OK: test_adicionar_planta_valida\n");
}

/* ... mais testes ... */

int main() {
    test_adicionar_planta_valida();
    /* chamar os restantes testes */
    return 0;
}
```

Tarefa 3: Testes de integração (0,8 valores)

a) Crie pelo menos 2 testes de integração que validem a interação entre módulos (um deve obrigatoriamente validar a persistência de dados e o outro a lógica de negócio entre módulos). Os cenários devem ser sobre situações relevantes do seu sistema;

b) Para cada teste de integração, inclua no relatório:
   - Cenário testado e módulos envolvidos;
   - Resultado esperado vs. resultado obtido;
   - Se o teste revelou algum problema de integração (e como foi corrigido).

No relatório, responda obrigatoriamente às seguintes questões de reflexão:

c) Identifique uma função no seu sistema que seria difícil de testar de forma isolada. Porquê? O que mudaria na sua arquitetura para a tornar mais testável?

d) O seu sistema reage a um CSV corrompido ou inexistente de que forma? Um teste poderia detectar esse problema? Como o escreveria?

Importante: é esperado que pelo menos um dos testes (unidade ou integração) falhe inicialmente e origine uma correção ao código. Descreva esse processo no relatório.

Se, excecionalmente, todos os seus testes passarem à primeira, apresente, em alternativa, um cenário hipotético, mas plausível, de falha (escolhendo uma função do sistema e descrevendo um possível erro, o teste que o detetaria e a correção que aplicaria).

Tarefa 4: Relatório final (1,0 valor)

Elabore um relatório (máx. 10-12 páginas, exceto capa e código desenvolvido) com a seguinte estrutura:

- Introdução: objetivo do trabalho e abordagem adotada;
- Tarefa 1: Melhorias de legibilidade: principais alterações (com exemplos antes/depois), decisões de documentação de *interfaces*;
- Tarefa 2: Testes de unidade: tabela com funções testadas, objetivos, relevância e resultados; exemplos de código de teste;
- Tarefa 3: Testes de integração: cenários testados e resultados; problemas encontrados e soluções;
- Conclusão: reflexão sobre a importância da legibilidade e dos testes; impacto no desenvolvimento profissional; aprendizagens e desafios;
- Anexos: código completo desenvolvido para o EfólioB, dentro do relatório e em formato editável.

Especificações de implementação

Organização e estrutura dos testes:

No desenvolvimento dos testes, deverá garantir a separação entre o código de produção e o código de verificação:

- Independência: Os testes de unidade e de integração não devem estar inseridos no código principal (main.c) do GreenTrack. Devem ser criados ficheiros de teste separados (ex: testes_unidade.c e testes_integracao.c), cada um com o seu próprio ponto de entrada (main), ou geridos através de diretivas de compilação;

- Automatização: Utilize o Makefile para criar alvos (*targets*) específicos para os testes (ver RF 4.3), que permitirá validar a robustez do sistema sem interferir com o executável final do utilizador;

- Foco dos testes: Enquanto que nos testes de unidade o objetivo é isolar uma função e testar os seus limites, nos testes de integração o foco deve estar na comunicação entre módulos.

O trabalho deve evidenciar o processo de desenvolvimento:

- Evidência de falha e correção: É obrigatório que o relatório documente detalhadamente os casos onde os testes (unidade ou integração) tenham falhado inicialmente, originando uma correção ou ajuste no código. Nos testes de unidade, inclua *printscreen* (capturas de ecrã) do terminal com o *output* do *assert*. Nos testes de integração, descreva a discrepância observada (neste caso o *printscreen* pode ser substituído pela descrição do estado incorreto detectado, por exemplo, o conteúdo do ficheiro, *output* do programa);

- Justificação de valores críticos: Na descrição dos testes de unidade, deve justificar tecnicamente a escolha de cada valor de entrada (*inputs*), explicando que limite ou cenário específico (normal, limite ou erro) pretendia validar;

- Verificação de execução: O relatório deve incluir uma captura de ecrã do terminal após a execução do comando *make test*, demonstrando que o conjunto automatizado de testes foi executado com sucesso no ambiente de desenvolvimento;

- Nexo causal: As melhorias de legibilidade descritas na Tarefa 1 devem estar diretamente relacionadas com o código entregue no E-fólio A, evitando refatorações genéricas sem contexto.

Critérios de avaliação e cotação (total 4 valores)

Critério	Componentes	Cotação
Legibilidade e documentação	Melhorias efetivas de legibilidade (0,4) + Documentação de *interfaces* completa (0,4) + Justificação de decisões e alternativas no relatório (0,4) Não serão aceites justificações genéricas como "ficou mais legível": deve explicar o impacto específico no seu projeto.	1,2
Testes de unidade	Cobertura adequada e diversidade de casos (caso normal, limite e erro) (0,3) + Qualidade e pertinência dos testes (0,3) + Uso correto de assert/printf com evidência de falha em *printscreen* (0,2) + Análise dos resultados e reflexão sobre sensibilidade dos valores de teste, incluindo a questão N±1 (0,2)	1,0
Testes de integração	Pertinência dos cenários escolhidos (0,2) + Implementação correta (0,3) + Análise, reflexão e resposta fundamentada (a resposta deve referenciar código concreto do sistema entregue, não princípios genéricos) às duas questões de reflexão obrigatórias (0,3)	0,8
Relatório	Clareza e estrutura (0,4) + Profundidade da reflexão crítica (reflexão profunda: referencia o seu código específico e antecipa implicações; superficial: descreve o que foi feito sem analisar o porquê) (0,4) + Cumprimento de formato (0,2). A ausência de evidências visuais de execução (*printscreens*) ou de justificações técnicas para os *inputs* de teste resultará numa penalização de até 50% neste critério	1,0

Normas a respeitar

- O código tem de compilar de modo a poder ser avaliado.
- Deve entregar dois ficheiros:
  - relatorio_NNNNNN_efolioB.pdf (relatório com todo o código desenvolvido para o Efólio B)
  - codigo_NNNNNN_efolioB.zip deve conter: o código fonte completo do GreenTrack melhorado (incluindo os ficheiros .h e .c de produção), os ficheiros de teste separados (testes_unidade.c, testes_integracao.c), o Makefile e o executável para Windows.
- Nomeie os ficheiros com o seu número de estudante (substituir NNNNNN).
- Evite a entrega próximo da hora limite.
- Não serão aceites ficheiros fora do prazo.
- Só serão aceites trabalhos submetidos na atividade criada no espaço de turma;
- Não serão aceites trabalhos com indícios de plágio.

Diretrizes para o relatório

- Fonte: Arial ou Times New Roman, tamanho 11
- Espaçamento: 1,5 entre linhas
- Margens: 2 cm em todos os lados
- Formato: PDF (preferencial) ou DOCX
- Máximo: 10-12 páginas (excluindo capa e código desenvolvido)

Votos de bom trabalho!