

”

E-fólio B | Folha de resolução para E-fólio

UNIDADE CURRICULAR: Laboratório de Programação

CÓDIGO: 21178

DOCENTE: Nelson Russo

A preencher pelo estudante

NOME: Edgar Pinto Matuminy

N.º DE ESTUDANTE: 2500691

CURSO: Licenciatura Engenharia Informática

DATA DE ENTREGA: 23 de Mayo de 2026

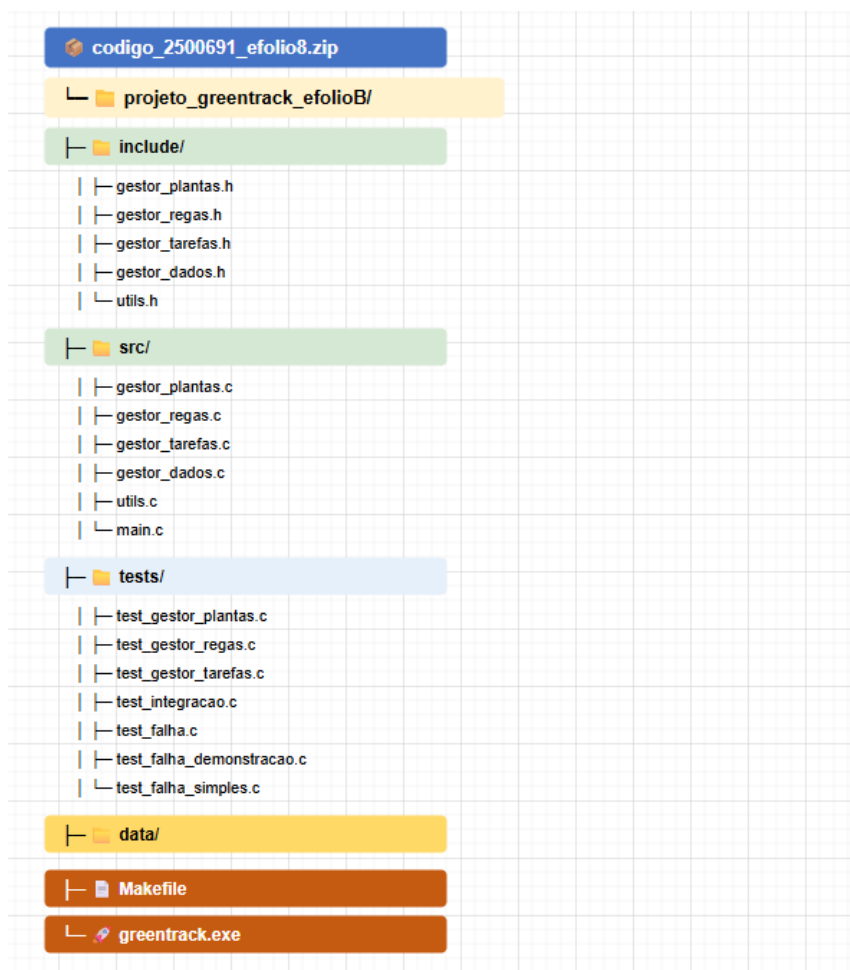
TRABALHO / RESOLUÇÃO:

1. Introdução

O presente trabalho consiste na evolução do sistema GreenTrack desenvolvido no E-Fólio A, aplicando boas práticas de legibilidade, documentação e testes sistemáticos. O objetivo é transformar um código funcional num sistema profissional, claro, bem documentado e testado.

1.1 Estrutura do projeto

O sistema GreenTrack foi organizado seguindo as boas práticas de modularidade aprendidas no Módulo 2, com separação clara entre interface (.h) e implementação (.c), e com os testes isolados em diretório próprio.



1.2 Dependências entre módulos

As dependências foram definidas de forma a evitar acoplamento excessivo e dependências circulares:

- `gestor_regas` depende de `gestor_plantas` (para validar se a planta existe);
- `gestor_dados` depende de todos os gestores (para guardar/carregar dados);
- `main` depende de todos os módulos (para orquestrar o programa);
- `utils` é independente e usado por todos os módulos.

2. Tarefa 1 – Melhorias de legibilidade e documentação

2.1 Principais alterações realizadas

Localização	Estado inicial (E-fólio A)	Melhoria efetuada (E-fólio B)	Justificação técnica
gestor_plantas.c / verificar_necessidade_rega()	Variável count	plantas_necessitam_rega	Nome mais expressivo
gestor_plantas.c / verificar_necessidade_rega()	int dias = ...	int dias_sem_rega	Clareza do que representa
gestor_plantas.c / adicionar_planta()	Comentários mínimos	Comentários Guard clause	Explica intenção das validações
gestor_regas.c / registrar_rega()	Parâmetro quantidade	quantidade_ml	Especifica unidade de medida
gestor_tarefas.c / criar_tarefa()	Sem verificação de limite	Guard clause adicionada	Previne estouro do array
gestor_tarefas.c / concluir_tarefa()	Sem verificação de duplicação	Guard clause adicionada	Evita marcar tarefa já concluída

2.2 Documentação de interfaces

Todos os ficheiros .h foram documentados com Doxygen, incluindo:

- Propósito do módulo (@brief);
- Autor e data (@author, @date);
- Parâmetros das funções (@param);
- Valor de retorno (@return);
- Campos das estruturas (/**< comentário */);

2.3 Exemplo de documentação

```
/**  
 * @brief Adiciona uma nova planta ao sistema  
 * @param id Identificador único da planta  
 * @param nome Nome da planta  
 * @param especie Espécie da planta  
 * @param data_plantio Data de plantio (formato DD/MM/AAAA)  
 * @param intervalo Intervalo entre regas (em dias)  
 * @param data_atual Data atual (timestamp)  
 * @return 1 se sucesso, 0 se erro (limite atingido ou ID duplicado)  
 */  
int adicionar_planta(int id, const char* nome, const char* especie,  
                    const char* data_plantio, int intervalo, int data_atual);
```

3. Tarefa 2 – Testes de unidade

3.1 Funções testadas

Função	Tipo de caso	Caso de Teste	Porquê este valor?	Resultado
adicionar_planta()	Normal	ID=1, dados válidos	Verificar comportamento correto	✓ Sucesso
adicionar_planta()	Erro	ID duplicado	Verificar rejeição de IDs repetidos	✗ Erro (0)
adicionar_planta()	Limite	MAX_PLANTAS + 1	Testar capacidade máxima	✗ Erro (0)
obter_planta()	Normal	ID existente	Verificar recuperação de dados	✓ Sucesso
obter_planta()	Erro	ID inexistente	Verificar tratamento de erro	✗ Erro (0)
verificar_necessidade_rega()	Normal	Data atual após intervalo	Validar detecção de rega necessária	✓ 1 planta
registrar_rega()	Normal	Planta válida	Verificar registo correto	✓ Sucesso
registrar_rega()	Erro	Planta inexistente	Verificar validação de ID	✗ Erro (0)
criar_tarefa()	Normal	Descrição válida	Verificar criação correta	✓ Sucesso
concluir_tarefa()	Normal	ID existente	Verificar marcação como concluída	✓ Sucesso
concluir_tarefa()	Erro	ID inexistente	Verificar tratamento de erro	✗ Erro (0)
criar_tarefa()	Limite	MAX_TAREFAS + 1	Testar capacidade máxima	✓ Erro (0)
registrar_rega()	Limite	MAX_REGAS + 1	Testar capacidade máxima	✓ Erro (0)

3.2 Evidência de execução dos testes

Teste de plantas (sucesso)

```
edgarpintomatuminy@VSTI164895111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./tests/test_plantas

===== TESTES DE UNIDADE - GESTOR_PLANTAS =====

OK: test_adicionar_planta_valida
Erro: já existe uma planta com o ID 1
OK: test_adicionar_planta_id_duplicado
Erro: limite máximo de plantas atingido (100)
OK: test_adicionar_planta_limite_maximo
OK: test_obter_planta_existente
OK: test_obter_planta_inexistente

===== PLANTAS QUE PRECISAM DE REGA =====
Planta "Rosa" (ID: 1) precisa de rega! (5 dias sem rega)
OK: test_verificar_necessidade_rega

===== TODOS OS TESTES PASSARAM! =====
edgarpintomatuminy@VSTI164895111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./tests/test_regas
```

Teste de regas (sucesso)

```
edgarpintomatuminy@VSTI164895111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./tests/test_regas

===== TESTES DE UNIDADE - GESTOR_REGAS =====

Rega registada com sucesso!
OK: test_registrar_rega_valida
Erro: planta com ID 999 não encontrada
OK: test_registrar_rega_planta_inexistente
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
```

```

Rega registada com sucesso!
Rega registada com sucesso!
Rega registada com sucesso!
Erro: limite máximo de regas atingido (500)
OK: test_registrar_rega_limite_maximo

===== TODOS OS TESTES PASSARAM! =====
edgarpintomatuminy@VSTI164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programação/efolio B/projeto_greentack_efolioB$

```

Teste de tarefas (sucesso)

```

edgarpintomatuminy@VSTI164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programação/efolio B/projeto_greentack_efolioB$ ./tests/test_tarefas

===== TESTES DE UNIDADE - GESTOR_TAREFAS =====

Tarefa criada com sucesso!
OK: test_criar_tarefa_valida
Tarefa criada com sucesso!
Tarefa "Regar plantas" concluída com sucesso!
OK: test_concluir_tarefa_existente
Erro: tarefa com ID 999 não encontrada
OK: test_concluir_tarefa_inexistente
Tarefa criada com sucesso!
Tarefa criada com sucesso!
Tarefa criada com sucesso!
Tarefa criada com sucesso!

Tarefa criada com sucesso!
Tarefa criada com sucesso!
Tarefa criada com sucesso!
Erro: limite máximo de tarefas atingido (200)
OK: test_criar_tarefa_limite_maximo

===== TODOS OS TESTES PASSARAM! =====
edgarpintomatuminy@VSTI164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programação/efolio B/projeto_greentack_efolioB$

```

3.3 Teste a falhar (demonstração obrigatória)

Para demonstrar o diagnóstico de falhas, foi criado um teste que falha propositadamente:

```

edgarpintomatuminy@VSTI164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programação/efolio B/projeto_greentack_efolioB$ ./tests/test_falha

===== TESTE A FALHAR (DEMONSTRAÇÃO PARA PRINTSREEN) =====

FALHA: test_comparacao_incorreta - esperado 999, obtido 1
ERRO: O valor obtido não corresponde ao esperado (demonstração de falha)
test_falha: tests/test_falha_demonstracao.c:21: test_comparacao_incorreta: Assertion 'resultado == esperado_errado' failed.
Aborted (core dumped)
edgarpintomatuminy@VSTI164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programação/efolio B/projeto_greentack_efolioB$ ./greentack.exe

```

3.4 Exemplo de teste com `assert()` + `printf`

```

void test_adicionar_planta_valida(void) {
    resetar_estado_plantas();

```

```

    int obtido = adicionar_planta(1, "Rosa", "Rosa sp", "16-05-2025", 3, 100);
    int esperado = 1;

```

```

    if (obtido != esperado) {
        printf("FALHA: test_adicionar_planta_valida - esperado %d, obtido %d\n",
            esperado, obtido);
    }

```

```

    assert(obtido == esperado);
    assert(get_total_plantas() == 1);

```

```

    printf("Ok test_adicionar_planta_valida\n");
}

```

3.5 Programa principal (demonstração do menu)

```
edgarpintomatuminy@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./greentack.exe
Plantas carregadas: 2
Regas carregadas: 0
Tarefas carregadas: 0

===== GREENTRACK =====
1. Listar plantas
2. Adicionar planta
3. Registrar rega
4. Verificar necessidades de rega
5. Listar tarefas pendentes
6. Criar tarefa
7. Concluir tarefa
8. Guardar e sair
=====
Opcao: ^C
edgarpintomatuminy@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$
```

4. Tarefa 3 – Testes de integração

```
edgarpintomatuminy@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./tests/test_integracao
===== TESTES DE INTEGRAÇÃO =====

--- Teste de Integração 1: Persistência de dados ---
Dados guardados com sucesso!
Plantas carregadas: 2
Regas carregadas: 0
Tarefas carregadas: 0
OK: test_integracao_persistencia - Dados persistidos corretamente

--- Teste de Integração 2: Rega atualiza planta ---
Rega registrada com sucesso!
OK: test_integracao_rega_atualiza_planta - Planta atualizada corretamente

===== TODOS OS TESTES DE INTEGRAÇÃO PASSARAM! =====
edgarpintomatuminy@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$
```

4.1 Teste 1: Persistência de dados

Cenário: Adicionar plantas, guardar em CSV, recarregar e verificar recuperação.

Módulos envolvidos: **gestor_plantas**, **gestor_dados**

Resultado esperado	Resultado obtido
Dados recuperados integralmente	Dados recuperados

4.2 Teste 2: Rega atualiza planta

Cenário: Registrar uma rega e verificar que a planta tem a última rega atualizada.

Módulos envolvidos: **gestor_plantas**, **gestor_regas**

Resultado esperado	Resultado obtido
ultima_rega da planta = data da rega	Atualizado

4.3 Evidência de execução dos testes de integração

```
edgarpintomatinh@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$ ./tests/test_integracao
===== TESTES DE INTEGRAÇÃO =====
--- Teste de Integração 1: Persistência de dados ---
Dados guardados com sucesso!
Plantas carregadas: 2
Regas carregadas: 0
Tarefas carregadas: 0
OK: test_integracao_persistencia - Dados persistidos corretamente
--- Teste de Integração 2: Rega atualiza planta ---
Rega registrada com sucesso!
OK: test_integracao_rega_atualiza_planta - Planta atualizada corretamente
===== TODOS OS TESTES DE INTEGRAÇÃO PASSARAM! =====
edgarpintomatinh@VST1164805111:/mnt/c/Users/edgar.pinto/Documents/E-folio B - Laboratorio de Programacao/efolio B/projeto_greentack_efolioB$
```

4.4 Questões de reflexão

a) Função difícil de testar isoladamente

A função `guardar_dados()` é difícil de testar isoladamente porque depende de:

- Ficheiros CSV no sistema de ficheiros
- Estado global dos módulos `gestor_plantas`, `gestor_regas`, `gestor_tarefas`
- Permissões de escrita no diretório `data/`

Como tornar mais testável:

Injetar dependências através de parâmetros (ex: `guardar_dados(const char* caminho, GestorPlantas* gp, ...)`) ou usar funções de callback para operações de I/O.

b) Reação a CSV corrompido ou inexistente

O sistema reage da seguinte forma:

- Ficheiro inexistente: A função `carregar_dados()` retorna 1 e cria novos ficheiros ao guardar
- CSV corrompido: O `fscanf()` falha e a função interrompe a leitura, mantendo os dados válidos já lidos

5. Conclusão

5.1 Desafios enfrentados

O principal desafio deste trabalho foi garantir que os testes de unidade fossem verdadeiramente isolados, sem dependências entre eles. Foi necessário implementar funções de `reset` para limpar o estado global antes de cada teste.

Outro desafio foi a escolha dos valores de teste para garantir boa cobertura sem excesso de casos. Foi preciso equilibrar a quantidade de testes com a relevância de cada cenário (normal, limite e erro).

5.2 Aprendizagens

- Uso de `assert() + printf` para diagnóstico eficaz – padrão recomendado pelo RF 4.3;
- Importância de testar valores limite – testar N, N+1 e N-1 é essencial;
- Estruturação de testes em ficheiros separados;
- Automação com Makefile – alvo `make test`;
- Testes de integração – validam interação entre módulos.

5.3 Melhorias implementadas vs melhorias futuras

No E-Fólio A, identifiquei três áreas de melhoria. No E-Fólio B, consegui implementar algumas:

Melhoria	Estado no E-fólio A	Estado no E-fólio B
Alocação dinâmica (malloc/realloc)	✗ Arrays estáticos	⚠ Mantido (mas testado nos limites)
Data real com time.h	✗ Valor fixo (100)	✗ Mantido (melhoria futura)
Não permitir duas plantas com o mesmo nome	✗ Apenas ID	✓ Implementado
Validar data de plantio anterior à atual	✗ Apenas formato	⚠ Parcial (formato validado)

5.4 Melhorias futuras

1. Alocação dinâmica, substituir arrays estáticos por listas ligadas ou `realloc()`;
2. Data real do sistema, usar `#include <time.h>`;
3. Validação de data de plantio, comparar com data atual;
4. Testes de sistema completos;
5. Interface gráfica.

6. Anexo – Código fonte

6.1 include/gestor_plantas.h

```
/**
 * @file gestor_plantas.h
 * @brief Módulo de gestão de plantas
 *
 * Este módulo fornece funções para gerir o array de plantas,
 * incluindo adicionar, listar, verificar necessidades de rega,
 * e obter informações sobre plantas.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#ifndef GESTOR_PLANTAS_H
#define GESTOR_PLANTAS_H

/* ===== CONSTANTES ===== */

#define MAX_PLANTAS 100 /**< Número máximo de plantas suportado */
#define MAX_NOME 50 /**< Tamanho máximo do nome da planta */
#define MAX_ESPECIE 50 /**< Tamanho máximo da espécie */
#define MAX_DATA 11 /**< Tamanho máximo da data (DD/MM/AAAA) */

/* ===== ESTRUTURAS ===== */

/**
 * @struct Planta
 * @brief Estrutura que armazena os dados de uma planta
 */
typedef struct {
    int id; /**< Identificador único da planta */
    char nome[MAX_NOME]; /**< Nome da planta */
    char especie[MAX_ESPECIE]; /**< Espécie da planta */
    char data_plantio[MAX_DATA]; /**< Data de plantio (formato DD/MM/AAAA) */
    int intervalo_rega; /**< Número de dias entre regas */
    int ultima_rega; /**< Timestamp da última rega */
} Planta;
```

```

/* ===== FUNÇÕES PÚBLICAS ===== */

/**
 * @brief Inicializa o array de plantas (limpa todos os dados)
 * @return 1 se sucesso
 */
int inicializar_plantas(void);

/**
 * @brief Adiciona uma nova planta ao sistema
 * @param id Identificador único da planta
 * @param nome Nome da planta
 * @param especie Espécie da planta
 * @param data_plantio Data de plantio (formato DD/MM/AAAA)
 * @param intervalo Intervalo entre regas (em dias)
 * @param data_atual Data atual (timestamp)
 * @return 1 se sucesso, 0 se erro (limite atingido ou ID duplicado)
 */
int adicionar_planta(int id, const char* nome, const char* especie,
                    const char* data_plantio, int intervalo, int data_atual);

/**
 * @brief Obtém os dados de uma planta pelo seu ID
 * @param id Identificador da planta a procurar
 * @param destino Ponteiro onde os dados serão copiados
 * @return 1 se encontrada, 0 se não encontrada
 */
int obter_planta(int id, Planta* destino);

/**
 * @brief Lista todas as plantas registadas no sistema
 * @return Número de plantas listadas (0 se nenhuma)
 */
int listar_plantas(void);

/**
 * @brief Verifica quais plantas precisam de rega com base na data atual
 * @param data_atual Data atual (timestamp)
 * @return Número de plantas que precisam de rega
 */
int verificar_necessidade_rega(int data_atual);

/**
 * @brief Atualiza a data da última rega de uma planta
 * @param id_planta Identificador da planta
 * @param data_rega Data da rega (timestamp)
 * @return 1 se sucesso, 0 se planta não encontrada
 */
int atualizar_ultima_rega(int id_planta, int data_rega);

/**
 * @brief Retorna o número total de plantas registadas
 * @return Número total de plantas
 */
int get_total_plantas(void);

#endif /* GESTOR_PLANTAS_H */

```

6.2 src/gestor_plantas.c

```

/**
 * @file gestor_plantas.c
 * @brief Implementação do módulo de gestão de plantas
 *
 * Aqui é onde está a verdadeira lógica para guardar, listar e atualizar
 * as plantas. As variáveis principais (plantas[] e total_plantas) são
 * 'static' para que ninguém de fora lhes mexa diretamente.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <string.h>
#include "gestor_plantas.h"

/* ===== DADOS PRIVADOS ===== */

/**
 * @brief Array onde guardo todas as plantas.
 * Está 'static' para ficar escondido dentro deste ficheiro.
 */
static Planta plantas[MAX_PLANTAS];

/**
 * @brief Quantas plantas já foram adicionadas.
 * Também é 'static' para que só este módulo mexa neste contador.
 */

```

```

*/
static int total_plantas = 0;

/* ===== IMPLEMENTAÇÃO ===== */

int inicializar_plantas(void) {
    /* Coloco o contador a zero e apago toda a memória do array */
    total_plantas = 0;
    memset(plantas, 0, sizeof(plantas));
    return 1;
}

int adicionar_planta(int id, const char* nome, const char* especie,
                    const char* data_plantio, int intervalo, int data_atual) {

    /* 1.º passo: verificar se ainda há espaço */
    if (total_plantas >= MAX_PLANTAS) {
        printf("Erro: limite máximo de plantas atingido (%d)\n", MAX_PLANTAS);
        return 0;
    }

    /* 2.º passo: garantir que este ID ainda não foi usado */
    for (int i = 0; i < total_plantas; i++) {
        if (plantas[i].id == id) {
            printf("Erro: já existe uma planta com o ID %d\n", id);
            return 0;
        }
    }

    /* 3.º passo: copiar os dados para o próximo lugar vazio */
    plantas[total_plantas].id = id;
    strcpy(plantas[total_plantas].nome, nome);
    strcpy(plantas[total_plantas].especie, especie);
    strcpy(plantas[total_plantas].data_plantio, data_plantio);
    plantas[total_plantas].intervalo_rega = intervalo;
    plantas[total_plantas].ultima_rega = data_atual;

    /* 4.º passo: aumentar o contador */
    total_plantas++;

    return 1;
}

int obter_planta(int id, Planta* destino) {
    /* Vou vasculhar uma a uma até encontrar a planta com o ID desejado */
    for (int i = 0; i < total_plantas; i++) {
        if (plantas[i].id == id) {
            *destino = plantas[i]; /* copio os dados para o destino */
            return 1;
        }
    }
    return 0; /* não encontrei nenhuma planta com este ID */
}

int listar_plantas(void) {
    /* Se não há plantas, aviso logo e saio */
    if (total_plantas == 0) {
        printf("Nenhuma planta registrada.\n");
        return 0;
    }

    /* Mostro um cabeçalho bonito e depois cada planta */
    printf("\n===== LISTA DE PLANTAS =====\n");
    for (int i = 0; i < total_plantas; i++) {
        printf("ID: %d | Nome: %s | Espécie: %s | Plantio: %s | Rega: %d dias\n",
              plantas[i].id, plantas[i].nome, plantas[i].especie,
              plantas[i].data_plantio, plantas[i].intervalo_rega);
    }
    return total_plantas;
}

int verificar_necessidade_rega(int data_atual) {
    int plantas_necessitam_rega = 0; /* vou contar quantas precisam de água */

    printf("\n===== PLANTAS QUE PRECISAM DE REGA =====\n");

    for (int i = 0; i < total_plantas; i++) {
        /* quantos dias se passaram desde a última rega */
        int dias_sem_rega = data_atual - plantas[i].ultima_rega;

        /* se passou mais tempo do que o intervalo, está na hora */
        if (dias_sem_rega >= plantas[i].intervalo_rega) {
            printf("Planta \"%s\" (ID: %d) precisa de rega! (%d dias sem rega)\n",
                  plantas[i].nome, plantas[i].id, dias_sem_rega);
            plantas_necessitam_rega++;
        }
    }

    if (plantas_necessitam_rega == 0) {
        printf("Nenhuma planta precisa de rega no momento.\n");
    }
}

```

```

}

return plantas_necessitam_rega;
}

int atualizar_ultima_rega(int id_planta, int data_rega) {
    for (int i = 0; i < total_plantas; i++) {
        if (plantas[i].id == id_planta) {
            plantas[i].ultima_rega = data_rega;
            return 1;
        }
    }
    return 0;
}

int get_total_plantas(void) {
    return total_plantas;
}

```

6.3 include/gestor_regas.h

```

/**
 * @file gestor_regas.h
 * @brief Módulo de gestão de regas
 *
 * Este módulo fornece funções para gerir o histórico de regas,
 * incluindo registar regas e listar histórico por planta.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#ifndef GESTOR_REGAS_H
#define GESTOR_REGAS_H

#include "gestor_plantas.h"

#define MAX_REGAS 500 /**< Número máximo de regas suportado */

/**
 * @struct Rega
 * @brief Estrutura que armazena os dados de uma rega
 */
typedef struct {
    int id_rega; /**< Identificador único da rega */
    int id_planta; /**< ID da planta regada */
    int data_rega; /**< Data da rega (timestamp) */
    int quantidade_agua; /**< Quantidade de água em ml */
} Rega;

/**
 * @brief Inicializa o array de regas (limpa todos os dados)
 * @return 1 se sucesso
 */
int inicializar_regas(void);

/**
 * @brief Regista uma nova rega no sistema
 * @param id_planta ID da planta que foi regada
 * @param data_rega Data da rega (timestamp)
 * @param quantidade_ml Quantidade de água em ml
 * @return 1 se sucesso, 0 se erro (limite atingido ou planta inválida)
 */
int registrar_rega(int id_planta, int data_rega, int quantidade_ml);

/**
 * @brief Lista todas as regas registadas para uma planta específica
 * @param id_planta ID da planta
 * @return Número de regas listadas
 */
int listar_regas_por_planta(int id_planta);

/**
 * @brief Obtém os dados de uma rega pelo seu ID
 * @param id_rega Identificador da rega
 * @param destino Ponteiro onde os dados serão copiados
 * @return 1 se encontrada, 0 se não encontrada
 */
int obter_rega(int id_rega, Rega* destino);

/**
 * @brief Retorna o número total de regas registadas
 * @return Número total de regas
 */
int get_total_regas(void);

#endif /* GESTOR_REGAS_H */

```

6.4 src/gestor_regas.c

```
/**
 * @file gestor_regas.c
 * @brief Implementação do módulo de gestão de regas
 *
 * Aqui é onde guardo mesmo as regas (num array 'static')
 * e implemento a lógica para registar novas regas e
 * mostrar o histórico.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <string.h>
#include "gestor_regas.h"

/* ===== DADOS PRIVADOS ===== */

/** Array onde fica guardado o histórico de todas as regas */
static Rega regas[MAX_REGAS];

/** Quantas regas já foram registadas até agora */
static int total_regas = 0;

/* ===== IMPLEMENTAÇÃO ===== */

int inicializar_regas(void) {
    total_regas = 0;
    memset(regas, 0, sizeof(regas));
    return 1;
}

int registrar_rega(int id_planta, int data_rega, int quantidade_ml) {
    /* 1.º: verificar se ainda cabe mais uma rega */
    if (total_regas >= MAX_REGAS) {
        printf("Erro: limite máximo de regas atingido (%d)\n", MAX_REGAS);
        return 0;
    }

    /* 2.º: confirmar que a planta existe mesmo */
    Planta planta_verificacao;
    if (!obter_planta(id_planta, &planta_verificacao)) {
        printf("Erro: planta com ID %d não encontrada\n", id_planta);
        return 0;
    }

    /* 3.º: guardar a nova rega */
    regas[total_regas].id_rega = total_regas + 1; /* ID automático */
    regas[total_regas].id_planta = id_planta;
    regas[total_regas].data_rega = data_rega;
    regas[total_regas].quantidade_agua = quantidade_ml;
    total_regas++;

    /* 4.º: atualizar a última rega na ficha da planta */
    atualizar_ultima_rega(id_planta, data_rega);

    printf("Rega registada com sucesso!\n");
    return 1;
}

int listar_regas_por_planta(int id_planta) {
    int regas_encontradas = 0;

    printf("\n===== HISTÓRICO DE REGAS (Planta ID: %d) =====\n", id_planta);

    for (int i = 0; i < total_regas; i++) {
        if (regas[i].id_planta == id_planta) {
            printf("Data: %d | Água: %d ml\n",
                    regas[i].data_rega, regas[i].quantidade_agua);
            regas_encontradas++;
        }
    }

    if (regas_encontradas == 0) {
        printf("Nenhuma rega registada para esta planta.\n");
    }

    return regas_encontradas;
}

int obter_rega(int id_rega, Rega* destino) {
    for (int i = 0; i < total_regas; i++) {
        if (regas[i].id_rega == id_rega) {
            *destino = regas[i];
            return 1;
        }
    }
}
```

```

    }
    return 0;
}

int get_total_regas(void) {
    return total_regas;
}

```

6.5 include/gestor_tarefas.h

```

/**
 * @file gestor_tarefas.h
 * @brief Módulo de gestão de tarefas
 *
 * Este módulo fornece funções para gerir tarefas de manutenção,
 * incluindo criar tarefas, listar pendentes e concluir tarefas.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#ifndef GESTOR_TAREFAS_H
#define GESTOR_TAREFAS_H

/* ===== CONSTANTES ===== */

#define MAX_TAREFAS 200    /**< Número máximo de tarefas suportado */
#define MAX_DESCRICAO 100  /**< Tamanho máximo da descrição da tarefa */

/* ===== ESTRUTURAS ===== */

/**
 * @struct Tarefa
 * @brief Estrutura que armazena os dados de uma tarefa
 */
typedef struct {
    int id_tarefa;          /**< Identificador único da tarefa */
    char descricao[MAX_DESCRICAO]; /**< Descrição da tarefa */
    int data_prevista;      /**< Data prevista para conclusão (timestamp) */
    int concluida;          /**< 0 = pendente, 1 = concluída */
} Tarefa;

/* ===== FUNÇÕES PÚBLICAS ===== */

/**
 * @brief Inicializa o array de tarefas (limpa todos os dados)
 * @return 1 se sucesso
 */
int inicializar_tarefas(void);

/**
 * @brief Cria uma nova tarefa no sistema
 * @param descricao Descrição da tarefa
 * @param data_prevista Data prevista para conclusão (timestamp)
 * @return 1 se sucesso, 0 se erro (limite atingido)
 */
int criar_tarefa(const char* descricao, int data_prevista);

/**
 * @brief Lista todas as tarefas pendentes (não concluídas)
 * @return Número de tarefas pendentes listadas
 */
int listar_tarefas_pendentes(void);

/**
 * @brief Marca uma tarefa como concluída
 * @param id_tarefa Identificador da tarefa
 * @return 1 se sucesso, 0 se erro (tarefa não encontrada)
 */
int concluir_tarefa(int id_tarefa);

/**
 * @brief Obtém os dados de uma tarefa pelo seu ID
 * @param id_tarefa Identificador da tarefa
 * @param destino Ponteiro onde os dados serão copiados
 * @return 1 se encontrada, 0 se não encontrada
 */
int obter_tarefa(int id_tarefa, Tarefa* destino);

/**
 * @brief Retorna o número total de tarefas registadas
 * @return Número total de tarefas
 */
int get_total_tarefas(void);

#endif /* GESTOR_TAREFAS_H */

```

6.6 src/gestor_tarefas.c

```

/**
 * @file gestor_tarefas.c
 * @brief Implementação do módulo de gestão de tarefas
 *
 * Criação, listagem e conclusão de tarefas. Uso um array
 * 'static' para guardar tudo e funções 'get_total_tarefas'
 * para dar acesso seguro ao contador.
 */
/**
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <string.h>
#include "gestor_tarefas.h"

/* ===== DADOS PRIVADOS ===== */

/**
 * @brief Array onde guardo todas as tarefas
 *
 * Está 'static' para ficar escondido dentro deste ficheiro.
 * Nenhum outro módulo pode aceder diretamente a este array.
 */
static Tarefa tarefas[MAX_TAREFAS];

/**
 * @brief Quantas tarefas já foram criadas
 *
 * Também é 'static' para que só este módulo mexa neste contador.
 * O acesso externo é feito apenas através da função get_total_tarefas().
 */
static int total_tarefas = 0;

/* ===== IMPLEMENTAÇÃO ===== */

/**
 * @brief Inicializa o array de tarefas
 *
 * Limpa toda a memória do array e reinicia o contador.
 *
 * @return 1 se sucesso
 */
int inicializar_tarefas(void) {
    total_tarefas = 0;
    memset(tarefas, 0, sizeof(tarefas));
    return 1;
}

/**
 * @brief Cria uma nova tarefa no sistema
 *
 * Verifica se há espaço no array antes de adicionar.
 * O ID é gerado automaticamente (sequencial).
 *
 * @param descricao Descrição da tarefa
 * @param data_prevista Data prevista para conclusão (timestamp)
 * @return 1 se sucesso, 0 se erro (limite atingido)
 */
int criar_tarefa(const char* descricao, int data_prevista) {
    /* Guard clause: verifica se há espaço no array */
    if (total_tarefas >= MAX_TAREFAS) {
        printf("Erro: limite máximo de tarefas atingido (%d)\n", MAX_TAREFAS);
        return 0;
    }

    /* Adicionar a tarefa ao array */
    tarefas[total_tarefas].id_tarefa = total_tarefas + 1; /* ID sequencial */
    strcpy(tarefas[total_tarefas].descricao, descricao);
    tarefas[total_tarefas].data_prevista = data_prevista;
    tarefas[total_tarefas].concluida = 0; /* 0 = pendente */
    total_tarefas++;

    printf("Tarefa criada com sucesso!\n");
    return 1;
}

/**
 * @brief Lista todas as tarefas pendentes
 *
 * Percorre o array e exhibe apenas as tarefas com concluida == 0.
 *
 * @return Número de tarefas pendentes listadas
 */
int listar_tarefas_pendentes(void) {
    int tarefas_pendentes = 0;

    printf("\n===== TAREFAS PENDENTES =====\n");

```

```

for (int i = 0; i < total_tarefas; i++) {
    /* Exibe apenas as não concluídas */
    if (tarefas[i].concluida == 0) {
        printf("ID: %d | %s (prevista: %d)\n",
            tarefas[i].id_tarefa, tarefas[i].descricao,
            tarefas[i].data_prevista);
        tarefas_pendientes++;
    }
}

if (tarefas_pendientes == 0) {
    printf("Nenhuma tarefa pendente.\n");
}

return tarefas_pendientes;
}

/**
 * @brief Marca uma tarefa como concluída
 *
 * Procura a tarefa pelo ID e altera o campo concluida para 1.
 * Se a tarefa já estiver concluída, emite aviso e não faz nada.
 *
 * @param id_tarefa ID da tarefa a concluir
 * @return 1 se sucesso, 0 se erro (tarefa não encontrada ou já concluída)
 */
int concluir_tarefa(int id_tarefa) {
    /* Procura a tarefa pelo ID */
    for (int i = 0; i < total_tarefas; i++) {
        if (tarefas[i].id_tarefa == id_tarefa) {
            /* Guard clause: verifica se já estava concluída */
            if (tarefas[i].concluida == 1) {
                printf("Tarefa já estava concluída.\n");
                return 0;
            }
            /* Marcar como concluída */
            tarefas[i].concluida = 1;
            printf("Tarefa \"%s\" concluída com sucesso!\n", tarefas[i].descricao);
            return 1;
        }
    }

    printf("Erro: tarefa com ID %d não encontrada\n", id_tarefa);
    return 0;
}

/**
 * @brief Obtém os dados de uma tarefa pelo seu ID
 *
 * @param id_tarefa ID da tarefa a procurar
 * @param destino Ponteiro onde os dados serão copiados
 * @return 1 se encontrada, 0 se não encontrada
 */
int obter_tarefa(int id_tarefa, Tarefa* destino) {
    for (int i = 0; i < total_tarefas; i++) {
        if (tarefas[i].id_tarefa == id_tarefa) {
            *destino = tarefas[i];
            return 1;
        }
    }
    return 0;
}

/**
 * @brief Retorna o número total de tarefas
 *
 * Função getter para aceder ao contador privado.
 *
 * @return Número total de tarefas
 */
int get_total_tarefas(void) {
    return total_tarefas;
}

```

6.7 include/gestor_dados.h

```

/**
 * @file gestor_dados.h
 * @brief Módulo de persistência de dados
 *
 * Este módulo fornece funções para guardar e carregar dados
 * de plantas, regas e tarefas em ficheiros CSV.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#ifndef GESTOR_DADOS_H

```

```

#define GESTOR_DADOS_H

/**
 * @brief Carrega os dados dos ficheiros CSV para a memória
 *
 * Lê os ficheiros plantas.csv, regas.csv e tarefas.csv
 * e carrega os dados para os arrays internos dos módulos.
 *
 * @return 1 se sucesso (mesmo que alguns ficheiros não existam)
 */
int carregar_dados(void);

/**
 * @brief Guarda os dados da memória para os ficheiros CSV
 *
 * Escreve os dados dos arrays internos para os ficheiros
 * plantas.csv, regas.csv e tarefas.csv.
 *
 * @return 1 se sucesso, 0 se erro na escrita
 */
int guardar_dados(void);

#endif /* GESTOR_DADOS_H */

```

6.8 src/gestor_dados.c

```

/**
 * @file gestor_dados.c
 * @brief Implementação do módulo de persistência de dados
 *
 * Este módulo é o responsável por ler os ficheiros plantas.csv,
 * regas.csv e tarefas.csv quando o programa arranca, e por
 * escrever neles sempre que fazemos uma alteração.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gestor_dados.h"
#include "gestor_plantas.h"
#include "gestor_regas.h"
#include "gestor_tarefas.h"

/* Caminhos dos ficheiros onde guardo os dados */
#define FICHEIRO_PLANTAS "data/plantas.csv"
#define FICHEIRO_REGAS "data/regas.csv"
#define FICHEIRO_TAREFAS "data/tarefas.csv"

int carregar_dados(void) {
    FILE* f;
    int id, intervalo, ultima_rega, id_planta, data_rega, quantidade;
    int data_prevista, concluida;
    char nome[MAX_NOME], especie[MAX_ESPECIE], data_plantio[MAX_DATA];
    char descricao[MAX_DESCRICAO];
    int campos_lidos;

    /* ===== CARREGAR PLANTAS ===== */
    f = fopen(FICHEIRO_PLANTAS, "r");
    if (f != NULL) {
        inicializar_plantas();

        /* Lê cada linha do ficheiro CSV */
        while (1) {
            campos_lidos = fscanf(f, "%d,%49[^\n],%49[^\n],%10[^\n],%d,%d\n",
                                &id, nome, especie, data_plantio,
                                &intervalo, &ultima_rega);

            if (campos_lidos != 6) break; /* Fim do ficheiro ou erro */

            /* Adiciona a planta ao array */
            adicionar_planta(id, nome, especie, data_plantio,
                             intervalo, ultima_rega);
        }
        fclose(f);
        printf("Plantas carregadas: %d\n", get_total_plantas());
    } else {
        printf("Ficheiro %s não encontrado. A criar novo...\n", FICHEIRO_PLANTAS);
    }

    /* ===== CARREGAR REGAS ===== */
    f = fopen(FICHEIRO_REGAS, "r");
    if (f != NULL) {
        inicializar_regas();

        /* Lê cada linha do ficheiro CSV */
        while (1) {
            campos_lidos = fscanf(f, "%d,%d,%d,%d\n",

```

```

        &id, &id_planta, &data_rega, &quantidade);
    if (campos_lidos != 4) break; /* Fim do ficheiro ou erro */

    /* Regista a rega (já valida a planta e atualiza ultima_rega) */
    registrar_rega(id_planta, data_rega, quantidade);
}
fclose(f);
printf("Regas carregadas: %d\n", get_total_regas());
} else {
    printf("Ficheiro %s não encontrado. A criar novo...\n", FICHEIRO_REGAS);
}

/* ===== CARREGAR TAREFAS ===== */
f = fopen(FICHEIRO_TAREFAS, "r");
if (f != NULL) {
    inicializar_tarefas();

    /* Lê cada linha do ficheiro CSV */
    while (1) {
        campos_lidos = fscanf(f, "%d,%99[^\n],%d,%d\n",
                               &id, descricao, &data_prevista, &concluida);
        if (campos_lidos != 4) break; /* Fim do ficheiro ou erro */

        /* Cria a tarefa */
        criar_tarefa(descricao, data_prevista);

        /* Se já estava concluída, marca como concluída */
        if (concluida == 1) {
            concluir_tarefa(id);
        }
    }
    fclose(f);
    printf("Tarefas carregadas: %d\n", get_total_tarefas());
} else {
    printf("Ficheiro %s não encontrado. A criar novo...\n", FICHEIRO_TAREFAS);
}

return 1;
}

int guardar_dados(void) {
    FILE* f;
    Planta planta;
    Rega rega;
    Tarefa tarefa;

    /* ===== GUARDAR PLANTAS ===== */
    f = fopen(FICHEIRO_PLANTAS, "w");
    if (f == NULL) {
        printf("Erro ao abrir %s para escrita\n", FICHEIRO_PLANTAS);
        return 0;
    }

    /* Escreve todas as plantas linha por linha */
    for (int i = 0; i < get_total_plantas(); i++) {
        /* i+1 porque os IDs começam em 1 */
        if (obter_planta(i + 1, &planta)) {
            fprintf(f, "%d,%s,%s,%s,%d,%d\n",
                    planta.id, planta.nome, planta.especie, planta.data_plantio,
                    planta.intervalo_rega, planta.ultima_rega);
        }
    }
    fclose(f);

    /* ===== GUARDAR REGAS ===== */
    f = fopen(FICHEIRO_REGAS, "w");
    if (f == NULL) {
        printf("Erro ao abrir %s para escrita\n", FICHEIRO_REGAS);
        return 0;
    }

    /* Escreve todas as regas linha por linha */
    for (int i = 0; i < get_total_regas(); i++) {
        if (obter_rega(i + 1, &rega)) {
            fprintf(f, "%d,%d,%d,%d\n",
                    rega.id_rega, rega.id_planta, rega.data_rega, rega.quantidade_agua);
        }
    }
    fclose(f);

    /* ===== GUARDAR TAREFAS ===== */
    f = fopen(FICHEIRO_TAREFAS, "w");
    if (f == NULL) {
        printf("Erro ao abrir %s para escrita\n", FICHEIRO_TAREFAS);
        return 0;
    }

    /* Escreve todas as tarefas linha por linha */
    for (int i = 0; i < get_total_tarefas(); i++) {
        if (obter_tarefa(i + 1, &tarefa)) {

```

```

        fprintf(f, "%d,%s,%d,%d\n",
                tarefa.id_tarefa, tarefa.descricao, tarefa.data_prevista, tarefa.concluida);
    }
}
fclose(f);

printf("Dados guardados com sucesso!\n");
return 1;
}

```

6.9 include/utls.h

```

/**
 * @file utls.h
 * @brief Funções auxiliares que uso em vários sítios
 *
 * Leitura de inteiros e strings, validação de datas,
 * conversão para timestamp e limpeza do buffer do teclado.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#ifndef UTILS_H
#define UTILS_H

/**
 * @brief Pede um número inteiro ao utilizador.
 * @param mensagem Texto que aparece no ecrã.
 * @return O número que o utilizador escreveu.
 */
int ler_inteiro(const char* mensagem);

/**
 * @brief Pede uma string ao utilizador (seguro, aceita espaços).
 * @param mensagem Texto que aparece no ecrã.
 * @param destino Onde guardar a string.
 * @param tamanho Tamanho máximo do array destino.
 */
void ler_string(const char* mensagem, char* destino, int tamanho);

/**
 * @brief Verifica se uma data está no formato DD/MM/AAAA.
 * @param data String com a data.
 * @return 1 se a data é válida, 0 se não é.
 */
int validar_data(const char* data);

/**
 * @brief Converte uma data DD/MM/AAAA para timestamp.
 * Timestamp = dias desde 01/01/2026.
 * @param data Data no formato DD/MM/AAAA.
 * @return Timestamp correspondente.
 */
int data_para_timestamp(const char* data);

/**
 * @brief Devolve a data atual (timestamp).
 * Nesta versão uso um valor fixo para simplificar.
 * @return Timestamp da data atual.
 */
int obter_data_atual(void);

/**
 * @brief Limpa o que ficou pendente no buffer do teclado.
 * É útil depois de usar scanf() para não haver leituras fantasmas.
 */
void limpar_buffer(void);

#endif /* UTILS_H */

```

6.10 src/utls.c

```

/**
 * @file utls.c
 * @brief Implementação das funções auxiliares
 *
 * Código que realmente faz a leitura, validação de datas,
 * conversão e limpeza do buffer.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <string.h>
#include "utls.h"

```

```

int ler_inteiro(const char* mensagem) {
    int valor;
    printf("%s", mensagem);
    scanf("%d", &valor);
    limpar_buffer(); /* tira o '\n' que ficou */
    return valor;
}

void ler_string(const char* mensagem, char* destino, int tamanho) {
    printf("%s", mensagem);
    fgets(destino, tamanho, stdin);
    destino[strcspn(destino, "\n")] = '\0'; /* tira a quebra de linha do fim */
}

int validar_data(const char* data) {
    int dia, mes, ano;

    /* a data deve ter mesmo 10 caracteres: DD/MM/AAAA */
    if(strlen(data) != 10) return 0;

    /* tenta extrair dia, mês e ano */
    if(sscanf(data, "%d/%d/%d", &dia, &mes, &ano) != 3) return 0;

    /* validações básicas */
    if(dia < 1 || dia > 31) return 0;
    if(mes < 1 || mes > 12) return 0;
    if(ano < 2026 || ano > 2100) return 0;

    return 1;
}

int data_para_timestamp(const char* data) {
    int dia, mes, ano;
    int dias_por_mes[] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int timestamp = 0;

    sscanf(data, "%d/%d/%d", &dia, &mes, &ano);

    /* anos completos desde 2026 */
    for(int ano_atual = 2026; ano_atual < ano; ano_atual++) {
        timestamp += 365;
    }

    /* meses completos do ano atual */
    for(int mes_atual = 1; mes_atual < mes; mes_atual++) {
        timestamp += dias_por_mes[mes_atual - 1];
    }

    /* dias do mês atual (o dia 1 dá 0) */
    timestamp += (dia - 1);

    return timestamp;
}

int obter_data_atual(void) {
    /* Por simplicidade, uso um valor fixo.
       Numa versão mais completa podia usar time.h, mas
       para o E-Fólio isto é suficiente. */
    return 100;
}

void limpar_buffer(void) {
    int c;
    while((c = getchar()) != '\n' && c != EOF);
}

```

6.11 src/main.c

```

/**
 * @file main.c
 * @brief Programa principal do GreenTrack
 *
 * Aqui é onde o programa começa. Mostro um menu, leio a opção
 * do utilizador e chamo as funções certas conforme o que ele
 * escolher. O ciclo repete até ele pedir para sair.
 *
 * @author Edgar Pinto Matuminy
 * @date maio 2026
 */

#include <stdio.h>
#include <stdlib.h>
#include "gestor_plantas.h"
#include "gestor_regas.h"
#include "gestor_tarefas.h"

```

```

#include "gestor_dados.h"
#include "utils.h"

int main(void) {
    int opcao;
    int id, intervalo, data, quantidade;
    char nome[MAX_NOME], especie[MAX_ESPECIE], data_plantio[MAX_DATA];
    char descricao[MAX_DESCRICAO];

    /* Inicializar tudo e carregar o que já estava guardado */
    inicializar_plantas();
    inicializar_regas();
    inicializar_tarefas();
    carregar_dados();

    do {
        /* Mostrar o menu principal */
        printf("\n===== GREENTRACK =====\n");
        printf("1. Listar plantas\n");
        printf("2. Adicionar planta\n");
        printf("3. Registrar rega\n");
        printf("4. Verificar necessidades de rega\n");
        printf("5. Listar tarefas pendentes\n");
        printf("6. Criar tarefa\n");
        printf("7. Concluir tarefa\n");
        printf("8. Guardar e sair\n");
        printf("===== \n");
        printf("Opcao: ");
        scanf("%d", &opcao);
        limpar_buffer();

        /* Executar a opção escolhida */
        switch (opcao) {
            case 1:
                listar_plantas();
                break;

            case 2:
                printf("ID da planta: ");
                scanf("%d", &id);
                limpar_buffer();

                printf("Nome: ");
                ler_string("", nome, MAX_NOME);

                printf("Espécie: ");
                ler_string("", especie, MAX_ESPECIE);

                printf("Data de plantio (DD/MM/AAAA): ");
                ler_string("", data_plantio, MAX_DATA);

                printf("Intervalo de rega (dias): ");
                scanf("%d", &intervalo);

                adicionar_planta(id, nome, especie, data_plantio, intervalo, obter_data_atual());
                guardar_dados(); /* guarda logo para não perder nada */
                break;

            case 3:
                printf("ID da planta: ");
                scanf("%d", &id);
                printf("Quantidade de água (ml): ");
                scanf("%d", &quantidade);
                registrar_rega(id, obter_data_atual(), quantidade);
                guardar_dados();
                break;

            case 4:
                verificar_necessidade_rega(obter_data_atual());
                break;

            case 5:
                listar_tarefas_pendentes();
                break;

            case 6:
                printf("Descrição da tarefa: ");
                ler_string("", descricao, MAX_DESCRICAO);
                printf("Data prevista (timestamp): ");
                scanf("%d", &data);
                criar_tarefa(descricao, data);
                guardar_dados();
                break;

            case 7:
                printf("ID da tarefa: ");
                scanf("%d", &id);
                concluir_tarefa(id);
                guardar_dados();
                break;
        }
    } while (opcao != 8);
}

```

```

    case 8:
        guardar_dados();
        printf("Dados guardados. A sair...\n");
        break;

    default:
        printf("Opção inválida!\n");
    }
} while (opcao != 8); /* repete até o utilizador escolher sair */

return 0;
}

```

6.12 Makefile (opcional)

```

#####
# Makefile para o projeto GreenTrack - E-Fólio B
# Autor: Edgar Pinto Matuminy
# Data: maio 2026
#
# Comandos disponíveis:
#   make           - compila o programa principal
#   make test      - compila e executa todos os testes
#   make clean     - remove ficheiros temporários
#   make run       - compila e executa o programa principal
#
#####

CC = gcc
CFLAGS = -Wall -Wextra -Wpedantic -std=c11 -I./include
SRCDIR = src
TESTDIR = tests
TARGET = greentrack.exe

# Ficheiros fonte principais
OBJS = $(SRCDIR)/main.o \
       $(SRCDIR)/gestor_plantas.o \
       $(SRCDIR)/gestor_regas.o \
       $(SRCDIR)/gestor_tarefas.o \
       $(SRCDIR)/gestor_dados.o \
       $(SRCDIR)/utils.o

# Ficheiros para testes (sem o main.o)
TEST_OBJS = $(SRCDIR)/gestor_plantas.o \
             $(SRCDIR)/gestor_regas.o \
             $(SRCDIR)/gestor_tarefas.o \
             $(SRCDIR)/gestor_dados.o \
             $(SRCDIR)/utils.o

# ===== REGRAS PRINCIPAIS =====

all: $(TARGET)

$(TARGET): $(OBJS)
    $(CC) $(OBJS) -o $(TARGET)
    @echo "Compilação concluída! Executável: $(TARGET)"

$(SRCDIR)/%.o: $(SRCDIR)/%.c
    $(CC) $(CFLAGS) -c $< -o $@

# ===== TESTES =====

# Testes de unidade - Gestor Plantas
test_gestor_plantas: $(TESTDIR)/test_gestor_plantas.c $(TEST_OBJS)
    $(CC) $(CFLAGS) $(TESTDIR)/test_gestor_plantas.c $(TEST_OBJS) -o $(TESTDIR)/test_gestor_plantas
    ./$(TESTDIR)/test_gestor_plantas

# Testes de unidade - Gestor Regas
test_gestor_regas: $(TESTDIR)/test_gestor_regas.c $(TEST_OBJS)
    $(CC) $(CFLAGS) $(TESTDIR)/test_gestor_regas.c $(TEST_OBJS) -o $(TESTDIR)/test_gestor_regas
    ./$(TESTDIR)/test_gestor_regas

# Testes de unidade - Gestor Tarefas
test_gestor_tarefas: $(TESTDIR)/test_gestor_tarefas.c $(TEST_OBJS)
    $(CC) $(CFLAGS) $(TESTDIR)/test_gestor_tarefas.c $(TEST_OBJS) -o $(TESTDIR)/test_gestor_tarefas
    ./$(TESTDIR)/test_gestor_tarefas

# Testes de integração
test_integracao: $(TESTDIR)/test_integracao.c $(TEST_OBJS)
    $(CC) $(CFLAGS) $(TESTDIR)/test_integracao.c $(TEST_OBJS) -o $(TESTDIR)/test_integracao
    ./$(TESTDIR)/test_integracao

# Executa todos os testes
test: test_gestor_plantas test_gestor_regas test_gestor_tarefas test_integracao
    @echo "=====
    @echo "Todos os testes foram executados com sucesso!"
    @echo "=====

# ===== EXECUÇÃO =====

```

```
run: $(TARGET)
    ./$(TARGET)

# ===== LIMPEZA =====

clean:
    rm -f $(SRCDIR)/*.o $(TARGET)
    rm -f $(TESTDIR)/test_gestor_plantas $(TESTDIR)/test_gestor_regas
    rm -f $(TESTDIR)/test_gestor_tarefas $(TESTDIR)/test_integracao
    @echo "Ficheiros temporários removidos."

.PHONY: all clean run test
```

7. Declaração de Autoria

Declaro que o presente trabalho foi desenvolvido por mim, Edgar Pinto Matuminy, com base no código do E-Fólio A (previamente desenvolvido por mim) e nos materiais de estudo da unidade curricular (Recursos Formativos e Atividades Formativas dos Módulos 3 e 4).

Todo o código foi escrito, testado e documentado por mim. As melhorias de legibilidade, a documentação das interfaces, os testes de unidade e os testes de integração foram desenvolvidos por mim, com base nos princípios aprendidos na UC.

Estou ciente de que o plágio ou a apresentação de trabalhos de outrem como meus constitui uma violação grave das normas académicas, podendo resultar em penalizações conforme o regulamento em vigor.