CEPEDI - EmbarcaTech

Tutorial: Orientação e Organização para o Projeto

2024

Tutorial: Orientação e Organização para o Projeto

Bem-vindos ao tutorial para manter a organização e colaborar no projeto **Conversor de Unidades**! Siga cuidadosamente as etapas abaixo para garantir que todos contribuam corretamente e que o trabalho flua sem problemas.

Orientações Gerais

1. Localização dos Arquivos:

- Coloque os códigos-fonte (**.c**) na pasta `src/`.
- Coloque os cabeçalhos (**.h**) na pasta `include/`.

2. Incluindo Arquivos de Cabeçalho:

- Use `#include "../include/nome_do_arquivo.h"` no início dos arquivos **.c** para importar as funções declaradas no cabeçalho.

3. Organização dos Arquivos:

- Cada membro deve criar apenas os arquivos que correspondem à sua tarefa.
- Não editem diretamente os arquivos criados por outros membros.

4. Exemplo de Estrutura Após Implementação:

```
conversor-unidades/
├── src/
│   ├── comprimento.c
│   ├── massa.c
│   ├── volume.c
│   ├── temperatura.c
│   └── ... (outros conversores)
├── include/
│   ├── comprimento.h
│   ├── massa.h
│   ├── volume.h
│   ├── temperatura.h
│   └── ... (outros cabeçalhos)
├── tests/ # Testes
│   ├── test_comprimento.c # Testes para comprimento
│   ├── test_massa.c # Testes para massa
│   ├── test_volume.c # Testes para volume
│   └── ... # Testes para outros conversores
├── main.c
└── README.md
```

5. Teste Local Antes do Pull Request:

- Compile com:

```
gcc -o conversor src/*.c -I include
```

Verifique se tudo funciona como esperado antes de abrir o Pull Request.

1. Clonar o Repositório

1.1. Certifique-se de que o Git está instalado em sua máquina.

- No terminal, digite:

```
git --version
```

- Se o comando não funcionar, instale o Git antes de continuar.

1.2. No terminal, clone o repositório do projeto:

```
git clone <URL_DO_REPOSITORIO>
```

- Substitua `<URL_DO_REPOSITORIO>` pelo link fornecido para o repositório no GitHub.

1.3. Acesse a pasta do projeto:

```
cd conversor-unidades
```

---

2. Entendendo a Estrutura do Projeto

O repositório já contém a seguinte estrutura:

```
conversor-unidades/
├── src/                    # Código-fonte (arquivos .c)
├── include/                # Cabeçalhos (arquivos .h)
├── tests/                  # Testes
├── README.md               # Documentação do projeto
```

- **Pasta `src/`:** Onde você colocará o arquivo `.c` referente à sua tarefa.
- **Pasta `include/`:** Onde você colocará o arquivo `.h` correspondente.
- **`README.md`:** Contém informações sobre o projeto.

---

3. Criar Sua Branch

Antes de começar a desenvolver, crie uma branch específica para sua tarefa. Isso ajuda a organizar o código e evita conflitos.

3.1. Confira se você está na branch principal (`main`):

```
git checkout main
```

3.2. Atualize o repositório para garantir que está com a versão mais recente:

```
git pull origin main
```

3.3. Crie e mude para uma nova branch com um nome descritivo:

```
git checkout -b feature/nome-da-tarefa
```

- Substitua `nome-da-tarefa` por algo relacionado à sua tarefa, como `conversor-comprimento` ou `conversor-massa`.

---

4. Desenvolvimento da Sua Tarefa

4.1. Navegue até a pasta `src/` e crie o arquivo `.c` correspondente à sua tarefa. Exemplo:

```
cd src
echo. > nome_tarefa.c
```

4.2. Volte para a pasta raiz e entre em `include/` para criar o arquivo `.h` correspondente:

```
cd ../include
echo. > nome_tarefa.h
```

4.3. Preencha os arquivos:

- No arquivo `.h`, declare as funções que você implementará no arquivo `.c`.
- No arquivo `.c`, implemente as funções.

---

5. Adicionar e Commitar as Alterações

Depois de concluir sua tarefa, siga estas etapas para salvar e preparar as alterações para envio ao repositório remoto:

5.1. Volte à pasta raiz do projeto:

```
cd ..
```

5.2. Adicione todas as alterações ao controle de versão:

```
git add .
```

5.3. Faça um commit descrevendo o que foi feito:

```
git commit -m "Implementação do conversor de [nome-da-tarefa]"
```

---

6. Enviar a Branch para o Repositório

Agora você deve enviar sua branch para o repositório remoto no GitHub.

6.1. Suba sua branch:

```
git push origin feature/nome-da-tarefa
```

6.2. Confirme que a branch foi enviada acessando o repositório no GitHub. Você verá sua branch listada.

---

7. Abrir um Pull Request

Depois de enviar sua branch, abra um **Pull Request (PR)** no GitHub para que o líder do grupo possa revisar seu código e integrar à branch principal.

7.1. Acesse o repositório no GitHub.
7.2. Clique na aba **Pull Requests**.
7.3. Clique em **New Pull Request**.
7.4. Selecione sua branch no lado direito e a branch `main` no lado esquerdo.
7.5. Adicione um título descritivo e um comentário sobre sua contribuição.
7.6. Clique em **Create Pull Request**.

Dicas Adicionais

- Nome dos Arquivos:
    - Use nomes coerentes e claros, como `comprimento.c` e `comprimento.h`.
- Padronização do Código:
    - Escreva comentários explicando cada função.
    - Siga as boas práticas discutidas na reunião.
- Revisão:
    - Antes de subir sua branch, revise seu código e faça testes para garantir que funciona corretamente.

8. Datas e Prazos

Para garantir o andamento do projeto, siga o cronograma abaixo:

- **Entrega Final do Projeto:** 26/12
    - **Prazo interno para concluir e subir as tarefas:** 22/12

      Todos os membros devem abrir um Pull Request no repositório até o dia 22/12, para que o líder possa revisar e integrar as contribuições à branch principal.

---

9. Tarefas e Nomes de Branch

Cada membro ficou responsável por uma tarefa específica. Abaixo estão as tarefas atribuídas, os responsáveis e as sugestões de nomes para as branches. Utilize o formato de nome indicado para criar sua branch no repositório.

Membro	Tarefa	Branch Sugerida
Matheus Mato	Criar conversor de comprimento	feature/conversor-comprimento
Vivian Rodrigues	Criar conversor de massa	feature/conversor-massa
Daniel Silva	Criar conversor de volume	feature/conversor-volume
Adimael Santos	Criar conversor de temperatura	feature/conversor-temperatura
Mychael Matos	Criar conversor de velocidade	feature/conversor-velocidade
Eric Franco	Criar conversor de potência	feature/conversor-potencia
Caio Bruno	Criar conversor de área	feature/conversor-area
Mychael Matos	Criar conversor de tempo	feature/conversor-tempo
Saulo	Criar conversor de dados	feature/conversor-dados
Tarefa Livre	Implementar Interface de usuário	
Tarefa Livre	Implementar testes e depuração	tests/implementacao
Adimael Santos	Criar o arquivo README.md	

Como Criar Sua Branch

Use o formato indicado em Branch Sugerida para criar sua branch. Exemplo para Matheus Mato:

```
git checkout -b feature/conversor-comprimento
```

Complete sua tarefa na branch criada, seguindo a estrutura do projeto explicada neste tutorial.

Siga as etapas para adicionar, commitar e enviar sua branch ao repositório remoto.

Abra um Pull Request com o nome e a descrição da sua tarefa para revisão.

10. Exemplo básico de implementação

Abaixo está um exemplo básico de como implementar um conversor no estilo sugerido para os arquivos **`nome_tarefa.c`** e **`nome_tarefa.h`**. Nesse exemplo, vamos usar o conversor de comprimento como base:

- Arquivo `comprimento.h`

Esse arquivo contém as declarações de funções e comentários explicativos sobre o que cada função faz.

```c
#ifndef COMPRIMENTO_H
#define COMPRIMENTO_H

/**
 * Converte metros para centímetros.
 *
 * @param metros O valor em metros.
 * @return O valor convertido em centímetros.
 */
double metros_para_centimetros(double metros);

/**
 * Converte metros para milímetros.
 *
 * @param metros O valor em metros.
 * @return O valor convertido em milímetros.
 */
double metros_para_milimetros(double metros);

/**
 * Converte centímetros para metros.
 *
 * @param centimetros O valor em centímetros.
 * @return O valor convertido em metros.
 */
double centimetros_para_metros(double centimetros);

#endif // COMPRIMENTO_H
```

- Arquivo `comprimento.c`

Esse arquivo contém as implementações das funções declaradas no .h.

```c
#include "../include/comprimento.h" // Inclui o cabeçalho correspondente

/**
 * Implementação da função que converte metros para centímetros.
 */
double metros_para_centimetros(double metros) {
    return metros * 100.0;
}

/**
 * Implementação da função que converte metros para milímetros.
 */
double metros_para_milimetros(double metros) {
    return metros * 1000.0;
}

/**
 * Implementação da função que converte centímetros para metros.
 */
double centimetros_para_metros(double centimetros) {
    return centimetros / 100.0;
}
```

11. Branch para Testes e Depuração

Nome da branch: `tests/implementacao`

- Isso deixa claro que a branch é dedicada aos testes e facilita o gerenciamento.
- Qualquer modificação ou adição de testes será realizada nesta branch.

Pasta no Repositório

No diretório do projeto, podemos criar uma pasta específica para os testes:

```
conversor-unidades/
├── src/                    # Código-fonte principal
│   ├── comprimento.c
│   ├── massa.c
│   ├── volume.c
│   └── ...
├── include/                # Arquivos de cabeçalho
│   ├── comprimento.h
│   ├── massa.h
│   ├── volume.h
│   └── ...
├── tests/                  # Testes
│   ├── test_comprimento.c # Testes para comprimento
│   ├── test_massa.c       # Testes para massa
│   ├── test_volume.c      # Testes para volume
│   └── ...                # Testes para outros conversores
├── main.c                  # Ponto de entrada da aplicação
└── README.md               # Documentação do projeto
```

Descrição dos Arquivos na Pasta `tests/`

1. **test_comprimento.c:**
    - Contém testes unitários para as funções implementadas em `comprimento.c`.
    - Exemplo: Verificar se a função `metros_para_centimetros` retorna o valor esperado para diferentes entradas.
2. **test_massa.c:**
    - Contém testes unitários para as funções implementadas em `massa.c`.
3. **test_volume.c**
    - Contém testes unitários para as funções implementadas em `volume.c`.

Biblioteca para Testes

Se possível, use uma biblioteca de testes como **Unity** (uma framework de teste para C) para organizar e executar os testes automaticamente.

Exemplo de Estrutura em `test_comprimento.c` com Unity:

```c
#include "unity.h"
#include "../include/comprimento.h"

void setUp() {
    // Inicialização necessária antes de cada teste
}

void tearDown() {
    // Limpeza após cada teste
}

void test_metros_para_centimetros() {
    TEST_ASSERT_EQUAL_DOUBLE(100.0, metros_para_centimetros(1.0));
    TEST_ASSERT_EQUAL_DOUBLE(250.0, metros_para_centimetros(2.5));
}

void test_centimetros_para_metros() {
    TEST_ASSERT_EQUAL_DOUBLE(1.0, centimetros_para_metros(100.0));
    TEST_ASSERT_EQUAL_DOUBLE(0.5, centimetros_para_metros(50.0));
}

int main() {
    UNITY_BEGIN();

    RUN_TEST(test_metros_para_centimetros);
    RUN_TEST(test_centimetros_para_metros);

    return UNITY_END();
}
```

Processo de Depuração

1. **Testar Localmente:**
    - Cada desenvolvedor pode executar os arquivos de teste para garantir que as funções estão corretas.
    - O comando comum para compilar e rodar testes é algo como:

```
gcc -o test_comprimento tests/test_comprimento.c src/comprimento.c
-Iinclude
./test_comprimento
```

2. **Correção de Erros:**
    - Se um teste falhar, revise o arquivo correspondente em `src/` ou `include/` e corrija os erros.
    - Recompilar e executar os testes novamente.
3. **Abrir Pull Request:**
    - Após concluir e verificar os testes, abra um Pull Request para integrar os arquivos de teste à branch `main`.

Vantagens de Seguir Esta Abordagem

- Facilita a depuração antes da integração.
- Mantém os arquivos de teste separados do código-fonte principal.
- Melhora a qualidade do projeto, garantindo que todas as funções sejam testadas.