Divisão das funcionalidades:

1. Unidades de comprimento (metro, centímetro, milímetro)	Augusto Barros ✅
2. Unidades de massa (quilograma, grama, tonelada)	Franklin Pereira ✅
3. Unidades de volume (litro, mililitro, metros cúbicos)	Gabriel Cerqueira ✅
4. Unidades de temperatura (Celsius, Fahrenheit, Kelvin)	Márcio Ruan ✅
5. Unidades de velocidade (km/h, m/s, mph)	Maryana Silveira ✅
6. Unidades de potência Watts (W), quilowatts (kW), cavalos-vapor (cv ou hp)	Mateus Silva ✅
7. Unidades de área (metro quadrado, centímetro quadrado)	Mateus Lima ✅
8. Unidades de tempo (segundos, minutos, horas)	Samuel Santos
9. Bits, bytes, kilobytes (KB), megabytes (MB), gigabytes (GB), terabytes (TB)	Sauan Santos ✅
10. Interface de usuário	Aline Machado ✅
11. Testes e **depuração**	Todos

Antes de qualquer coisa:

- Instalar o git no vscode
- Instalar a extensão Makefile Tools da Microsoft no vscode
- Criar uma conta no github e fazer o login nela.

Estrutura de pastas:

```
ConversorUnidadesC/
├── bin/ # Executáveis gerados
├── obj/ # Arquivos objeto (.o)
├── src/ # Arquivos-fonte (.c)
│   ├── main.c
│   ├── conversorComprimento.c
│   ├── conversorTemperatura.c
│   └── ...
├── include/ # Arquivos de cabeçalho (.h)
│   ├── conversorComprimento.h
│   ├── conversorTemperatura.h
│   └── ...
├── .gitignore
├── Makefile # Script de build
└── README.md
```

Depois que você fizer o git clone, você vai precisar criar em seu projeto local dois arquivos:

- .gitignore:

Colar o código abaixo dentro dele

```
.DS_Store
Thumbs.db


bin/
obj/

.vscode/
*.swp
*.swo
```

- Makefile

Colar esse código em seu Makefile:

```
CC = gcc
CFLAGS = -Iinclude -Wall -Wextra
SRCDIR = src
INCDIR = include
BINDIR = bin
OBJDIR = obj

SOURCES = $(wildcard $(SRCDIR)/*.c)
OBJECTS = $(patsubst $(SRCDIR)/%.c, $(OBJDIR)/%.o, $(SOURCES))
EXECUTABLE = $(BINDIR)/ConversorUnidades

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS) | $(BINDIR)
    $(CC) $(OBJECTS) -o $@

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

$(BINDIR) $(OBJDIR):
    mkdir -p $@

clean:
    rm -rf $(OBJDIR) $(BINDIR)

.PHONY: all clean
```

O nosso Makefile vai nos ajudar de forma rápida e prática a gerar todos os executáveis dentro da pasta /bin e os .o na pasta /obj independente de você ser usuário Linux ou Windows. Além disso, essa organização vai permitir que a gente possa usar o .gitignore para evitar que esses executáveis sejam enviados para o repositório remoto.

Para executar seu código você precisa ir no vscode (**DEPOIS QUE VOCÊ TIVER CLONADO PARA O REPOSITORIO LOCAL),** para a sua pasta raiz no terminal (ex: cd "/home/aline/embarcatech/unidade3/codigos/ConversorUnidadesC/") e digitar:

```
bash

make
```

Para executar seu código:

```
bash

./bin/ConversorUnidades
```

Fiz um exemplo com unidade de medida de energia para vocês saberem o que vai em cada arquivo. Observe os arquivos conversorEnergia.c e conversorEnergia.h. Se vocês quiserem podem usar eles como modelo para os de vocês. Aqui consegui executar normal.

Vejam que já deixei o main.c com o menu organizado com algumas partes comentadas para aguardar vocês fazerem seus códigos. Se quiserem testar é só descomentar as partes referentes aos códigos de vocês.

Fluxo de Trabalho com Git

1. Clonagem do Repositório:

Cada integrante deve clonar o repositório para sua máquina local. Então com o seu vscode aberto você vai digitar na linha de comando:

```
bash

git clone https://github.com/alinemach/ConversorUnidadesC.git
```

2. Criação de Branches:

Observe que já existem duas branches, uma *main* e a outra *develop*. Crie sua branch específica de acordo com sua funcionalidade ou correção. Por exemplo, se você está trabalhando com o conversor de comprimento você vai criar a feature da seguinte forma:

```
bash

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

3. Certifique-se de estar na branch correta

Você deve primeiro verificar em qual branch está e mudar para a branch feature/nome-da-feature que você vai trabalhar:

```
bash

# Verifica em qual branch está
git branch

# Caso não esteja na branch feature, muda para ela
git checkout feature/nome-da-feature
```

4. Crie as alterações no código

Vocês podem criar e editar os arquivos necessários e realizar as implementações.
Aqui é a hora de colocar a mão na massa e desenvolver seu código! ;)

5. Adicione os arquivos alterados

Depois de finalizar as alterações, o desenvolvedor deve adicionar os arquivos ao *staging area*:

```
bash

# Adiciona todos os arquivos modificados
git add .
```

6. Faça o commit das alterações

É importante que a mensagem do commit seja clara e descritiva, seguindo boas práticas:

```
bash

# Commit com mensagem descritiva
```

```
git commit -m "Implementa conversor de temperatura com suporte a Celsius,
Fahrenheit e Kelvin"
```

7. Envie a branch para o repositório remoto

Após o commit, a branch feature/nome-da-feature deve ser enviada para o repositório remoto para que você possa revisar e fazer o merge:

```
bash

# Envia a branch para o repositório remoto
git push origin feature/nome-da-feature
```

8. Crie um Pull Request (PR)

1. O desenvolvedor acessa o repositório no GitHub.
2. No GitHub, o desenvolvedor verá um botão sugerindo a criação de um Pull Request após o *push*.
3. O Pull Request deve ser configurado para mesclar a branch feature/nome-da-feature na branch develop.
4. O desenvolvedor deve adicionar uma descrição explicando o que foi implementado.

Assista ao vídeo no youtube sobre PR:
https://www.youtube.com/watch?v=y_koBLR8vfw

Comandos resumidos para a equipe

```
bash

git checkout feature/nome-da-feature     # Acessa a branch feature
git add .                                 # Adiciona os arquivos alterados
git commit -m "Descrição do commit"      # Realiza o commit
git push origin feature/nome-da-feature  # Envia a branch para o repositório remoto
```

OBS: Dica para organização

Certifique-se de que todos os desenvolvedores:

- Dêem nomes claros e consistentes às branches (por exemplo, feature/conversor-temperatura).
- Escrevam mensagens de commit detalhadas e relevantes.
- Atualizem a branch develop localmente antes de criar uma nova branch feature para evitar conflitos futuros:

```
bash

git checkout develop
git pull origin develop
```