Programação Imperativa e Funcional

Modularização e Compilação

Diego Bezerra

dfb2@cesar.school

C.E.S.A.R school

Nas aulas anteriores...

- Tipos Estruturados
- Listas de exercícios

Objetivos

- Introdução à modularização e compilação

Introdução a modularização

- À medida que um **programa cresce e fica mais complexo**, o número de funções e de linhas de código em **um único arquivo** pode fazer com que sua **compreensão e manutenção se torna mais difícil**
- Para facilitar, o compilador C permite a **separação do código em arquivos diferentes**, chamados “módulos”

Separação em módulos 1/2

- Recomenda-se criar um módulo, **separando a biblioteca de funções ou tipos** em dois arquivos
  - Arquivo de declarações ou arquivo cabeçalho (**header**) – **extensão *.h**: **declara** as funções existentes naquela biblioteca, porém não provê sua implementação. Declaram a interface (o "que")
  - Arquivo de definições/implementações ou arquivo de código – **extensão *.c**: define a **implementação** de todas as funções declaradas no arquivo cabeçalho. Implementam o comportamento (o "como")

Separação em módulos 2/2

- Caso a biblioteca de funções utilize **tipos de dados compostos/estruturados**, estes usualmente são definidos no arquivo cabeçalho (*.h).
- Tanto o arquivo da biblioteca de funções quanto o programa principal devem importar (**include**) o arquivo cabeçalho
  - Busca garantir que a função implementada na biblioteca e chamada no programa seja a mesma (mesmo nome, parâmetros e tipo de retorno)
- O include agora usa aspas duplas: **#include “header.h”**

Separação em módulos: Exemplo

```
// Arquivo operacoes.h
int soma(int a, int b);
```

```
// Arquivo operacoes.c
#include <stdio.h>
#include "operacoes.h"

int soma(int a, int b) {
    return a + b;
}
```

```
// Arquivo main.c
#include "operacoes.h"
int main() {
    soma(3, 5); return 0;
}
```

```
// Compilando via terminal
// Arquivo executavel
$ gcc *.c -o nome_executavel
```

Separação em módulos

- Mas não posso definir e implementar em um arquivo *.c e fazer include dele? Sim, até funciona, mas…
  - Duplicação de código pode gerar erros durante o processo de linking na compilação (**Multiple definition of function**)
- Problemas de visibilidade
- Fuga ao padrão de desenvolvimento

Boas práticas: include guard

- A estrutura **include guard** evita múltiplas inclusões do mesmo header durante a compilação, evitando erros de compilação, redefinições de variáveis, tipos ou funções

```
// Arquivo operacoes.h

#ifndef OPERACOES_H // verfica se OPERACOES_H nao esta definido
#define OPERACOES_H // se nao estiver, define OPERACOES_H

int soma(int a, int b);

#endif // OPERACOES_H // finaliza a condicional e encerra include guard
```

Boas práticas: diretórios

- Uma estrutura bem organizada facilita a navegação, manutenção e escalabilidade do código
- Um projeto pode ser organizado considerando a seguinte estrutura de diretórios

```
/meu_projeto
├── src/          # Código-fonte (.c)
├── include/      # Cabeçalhos (.h)
├── build/        # Saídas de compilação (não versionado)
├── tests/            # Testes unitários
├── docs/             # Documentação do projeto
├── Makefile          # Arquivo de build
├── README.md         # Descrição do projeto
└── LICENSE           # Licença
```

Boas práticas: diretórios

```
// Arquivo operacoes.h
// Dentro do diretorio include

#ifndef OPERACOES_H
#define OPERACOES_H

int soma(int a, int b);

#endif // OPERACOES_H
```

```
// Arquivo operacoes.c
// Dentro do diretorio src
#include <stdio.h>
#include "operacoes.h"

int soma(int a, int b) {
    return a + b;
}
```

```
// Arquivo main.c
// Dentro do diretorio src

#include "operacoes.h"
int main() {
    soma(3, 5); return 0;
}
```

```
// Compilando via terminal
// Dentro do diretorio build
$ gcc ../src/*.c -I ../include/ -o main
```

Bibioteca exemplo:
https://github.com/tgfb/cli-lib

Exercícios 1

- Crie um arquivo header para definir funções que realizam as quatro operações básicas da matemática entre dois números
- Defina a implementação dessas funções em um arquivo *.c
- Chame as funções na main, testando a corretude das operações
- Compile seu projeto

Exercícios 2

- Crie um arquivo header para definir structs que representam formas geométricas
- Defina a implementação desse funções que calculam a área dessas figuras um arquivo *.c
- Chame as funções em um arquivo que contém uma função main, testando a corretude das operações
- Compile seu projeto

Referências

- Rangel Netto, J. L. M., Cerqueira, R. D. G., & Celes Filho, W. (2004). **Introdução a estrutura de dados: com técnicas de programação em C**.