Organização de código

Video desta aula

À medida em que um software cresce em tamanho e funcionalidades, seu código-fonte deve ser organizado corretamente para facilitar sua compreensão, manutenção e evolução. É importante quebrar o código-fonte em arquivos separados, dividindo-o de acordo com os módulos e/ou funcionalidades do sistema.

O método usual de organização de código em C consiste em dividir o programa em módulos que são vistos como "bibliotecas", provendo funcionalidades para a construção do programa principal. O programa principal deve usar as funcionalidades desses módulos e permanecer o mais compacto e abstrato possível.

Um programa em C é estruturado em arquivos de código (extensão `.c`) e arquivos de cabeçalho (*header*, extensão `.h`). Os arquivos de código contêm implementações concretas, enquanto os de cabeçalho contêm protótipos de funções e os tipos de dados necessários a esses protótipos.

Dado um arquivo `cpx.c` contendo funções e tipos de dados para manipular números complexos, o arquivo `cpx.h` deve ser visto como a definição da **interface** para outros arquivos C usarem as funcionalidades implementadas por `cpx.c`.

Com isso, funções e tipos que são usados **somente** dentro de `cpx.c` não precisam (nem devem) aparecer no arquivo de interface `cpx.h`.

Ao dividir o código-fonte em arquivos separados, alguns cuidados devem ser tomados:

- Agrupe as funções e definições de dados associados ao mesmo tópico ou assunto em um mesmo arquivo `.c`;
- Coloque os protótipos das funções públicas e as definições de dados necessárias a esses protótipos em um arquivo de cabeçalho `.h` com o mesmo nome do arquivo `.c` correspondente;
- Somente faça inclusões (`#include`) de arquivos de cabeçalho (`.h`).

Maldições imperdoáveis  :

- fazer inclusão de arquivos "`.c`" (`#include "arquivo.c"`)
- colocar código real (for, if, while, ...) em arquivos de cabeçalho [1)]

Exemplo

Este exemplo implementa uma mini-biblioteca de números complexos, ou seja, um conjunto de funções para definir e manipular números complexos[2)].

O arquivo principal (neste exemplo, `exemplo.c`) usa funções dessa biblioteca. Para isso, ele deve incluir todos os arquivos de cabeçalho necessários para sua compilação e também deve definir a função `main`:

exemplo.c

```c
// Demonstraçao da biblioteca simples de números complexos :-)
// Carlos Maziero, DINF/UFPR 2020

#include <stdio.h>
#include "cpx.h"

int main ()
{
  cpx_t a, b, c, d ;

  // (10 + 7i) + (-2 + 4i) = (8 + 11i)
  a = cpx (10, 7) ;
  b = cpx (-2, 4) ;
  c = cpx_sum (a, b);
  printf ("c vale %s\n", cpx_str (c)) ;

  // (3 + 2i) * (1 + 4i) = –5 + 14i
  d = cpx_mul (cpx (3, 2), cpx (1, 4));
  printf ("d vale %s\n", cpx_str (d)) ;
}
```

Como o arquivo `exemplo.c` não define funções (ou estruturas, tipos, etc) que serão usadas em outros arquivos do programa, não se deve criar um arquivo `exemplo.h`.

Nossa “biblioteca” de números complexos é implementada pelos arquivos `cpx.c` e `cpx.h`.

O arquivo de cabeçalho `cpx.h` deve declarar somente informações públicas: os tipos de dados e protótipos das funções que devem ser conhecidas por quem irá utilizar as funcionalidades da biblioteca:

cpx.h

```c
// Biblioteca simples de números complexos :-)
// Carlos Maziero, DINF/UFPR 2020

#ifndef __CPX__
#define __CPX__

// estrutura de um número complexo
typedef struct {
  float r, i;   // componentes real e imaginária
} cpx_t ;

// define o valor de um complexo
cpx_t cpx (float r, float i) ;

// operações aritméticas entre dois complexos
cpx_t cpx_sum (cpx_t a, cpx_t b) ;
cpx_t cpx_sub (cpx_t a, cpx_t b) ;
cpx_t cpx_mul (cpx_t a, cpx_t b) ;
cpx_t cpx_div (cpx_t a, cpx_t b) ;

// devolve os componentes de um número complexo
float cpx_real (cpx_t c) ;
```

```
float cpx_imag (cpx_t c) ;

// gera uma string a partir de um número complexo
char* cpx_str (cpx_t c) ;

// outras operações
// ...

#endif
```

Deve-se observar o uso das macros de pré-compilação `#ifndef` e `#define`. Elas constituem uma *include guard*, usada para evitar a repetição das definições, caso o mesmo arquivo de cabeçalho seja incluído múltiplas vezes em diferentes locais do código.

Por sua vez, o arquivo `cpx.c` contém as informações privadas da biblioteca (estruturas de dados internas, variáveis globais) e as implementações das funções definidas em `cpx.h`. Esse arquivo deve incluir todos os cabeçalhos necessários à implementação das funções.

cpx.c

```
// Biblioteca simples de números complexos :-)
// Carlos Maziero, DINF/UFPR 2020

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "cpx.h"

// conversão de coordenadas polares em retangulares.
// Obs: esta é uma função interna; com o modificador "static",
// ela só pode ser acessada por código dentro deste arquivo.
static void polar_to_rect (float r, float a, float *x, float *y)
{
  // implementação da função
  // ...
}

// define os valores de um número complexo
cpx_t cpx (float r, float i)
{
  cpx_t new = {r, i} ;
  return (new) ;
}

// soma de dois complexos
cpx_t cpx_sum (cpx_t a, cpx_t b)
{
  cpx_t sum ;
  sum.r = a.r + b.r ;
  sum.i = a.i + b.i ;
  return (sum) ;
}

// diferença de dois complexos
cpx_t cpx_sub (cpx_t a, cpx_t b)
```

```
{
  cpx_t sum ;
  sum.r = a.r - b.r ;
  sum.i = a.i - b.i ;
  return (sum) ;
}

// produto de dois complexos
cpx_t cpx_mul (cpx_t a, cpx_t b)
{
  cpx_t prod ;
  prod.r = a.r * b.r - a.i * b.i ;
  prod.i = a.r * b.i + a.i * b.r ;
  return (prod) ;
}

// implementação das demais funções
// ...
```

Em resumo:

- `cpx.c`: implementação das funções de manipulação de números complexos.
- `cpx.h`: interface (protótipos) das funções **públicas** definidas em `cpx.c`.
- `exemplo.c`: programa principal, que usa as funções descritas em `cpx.h` e implementadas em `cpx.c`.

Para compilar:

```
cc -Wall exemplo.c cpx.c -o exemplo
```

O arquivo `cpx.c` também pode ser compilado separadamente, gerando um arquivo-objeto `cpx.o` que poderá ser ligado ao arquivo `exemplo.o` posteriormente:

```
cc -Wall -c cpx.c

cc -Wall exemplo.c cpx.o -o exemplo
```

Essa organização torna mais simples a construção de bibliotecas e a distribuição de código binário para incorporação em outros projetos (reuso de código). Além disso, essa estruturação agiliza a compilação de grandes projetos, através do sistema Make.

[1] Exceto quando se tratar de funções ''inline''

[2] Esta biblioteca é inútil, pois o padrão C99 já inclui o suporte a números complexos ...