Modularização em `C`

Daniel Saad Nogueira Nunes

1 Introdução

A linguagem `C` permite dividir as funcionalidades do sistema a ser construído em diversos módulos, sendo cada módulo responsável por realizar um tipo de tarefa. Estes módulos podem ser divididos em vários arquivos.

Cada módulo exporta um *contrato* para os outros módulos. Este contrato define as funcionalidades providas pelo módulo que está exportando tal contrato. Para utilizar as funcionalidades de um módulo, um segundo módulo não necessita saber como elas estão implementadas, apenas a assinatura de tais funcionalidades.

O emprego da divisão do sistema em módulos ajuda a isolar falhas e o desenvolvimento em equipes, uma vez que uma equipe não precisa saber como outro módulo está implementado, apenas conhecer as funcionalidades previstas por ele e utilizá-las para construir outra parte do sistema.

Pense na biblioteca padrão do `C`, até o momento você vem utilizando as funções `scanf` e `printf` sem necessariamente saber como elas estão implementadas. Contudo, para utilizá-las, você teve que indicar que queria utilizar o módulo `stdio.h`, que contém as assinaturas das funções de entrada/saída.

Veremos agora como dividir um sistema em módulos utilizando a linguagem `C`.

2 Modularização

Os arquivos em `C` podem ser basicamente de dois tipos: os arquivos cabeçalhos (`.h`) e os arquivos de implementação (`.c`).

Os arquivos cabeçalhos (*headers*), expõem o contrato do módulo. Eles contém as assinaturas das funções implementadas naquele módulo e outras definições básicas, como tipos e constantes inerentes ao módulo. Outro módulo pode importar este contrato com a diretiva `#include`.

Os arquivos de implementação por sua vez, contém a implementação de cada função presente no `.h`.

Vamos pegar o seguinte caso, queremos um módulo que possibilite operações sobre vetores. Teremos os seguintes módulos:

- Módulo de leitura de vetores. Responsável por ler vetores da entrada padrão.

- Módulo de impressão de vetores. Responsável por escrever vetores na saída padrão.
- Módulo de operação entre vetores. Responsável por efetuar operações entre vetores.

Para simplificar a discussão, vamos assumir que estamos considerando apenas vetores de inteiros (`int`).

O contrato do módulo de leitura poderia prover a função `le_vetor`, que recebe o vetor e o tamanho, e realiza a leitura do vetor. Isto poderia ser feito como no código abaixo.

```
1  #ifndef LEITURA_H
2  #define LEITURA_H
3
4  void le_vetor(int vetor[],int n);
5
6  #endif
```

Repare que apenas o protótipo da função está presente no arquivo cabeçalho.

As linhas 1 e 2 trazem uma coisa essencial, uma diretiva de inclusão condicional. Caso a macro `LEITURA_H` não esteja definida, ela é definida na linha 2 juntamente com o protótipo desta função. Desta forma, caso vários arquivos incluam este arquivo cabeçalho, não haverá definições múltiplas da função `le_vetor`, pois se ela já tiver sido definida da primeira vez, o `#ifndef` avaliará como falso, e não realizará a importação do arquivo novamente.

É boa prática de programação sempre colocar as diretivas de compilação condicional nos arquivos cabeçalhos e utilizar macros referentes ao nome do arquivo em maiúsculo e trocando o símbolo de espaço pelo underscore (_).

Agora que já definimos o contrato do módulo de leitura, podemos implementar suas funcionalidades no arquivo de implementação, conforme o código abaixo:

```
1  #include <stdio.h>
2  #include "leitura.h"
3
4  void le_vetor(int vetor[],int n){
5          int i;
6          for(i=0;i<n;i++){
7                  printf("vetor[%d] = ",i);
8                  scanf("%d",&vetor[i]);
9          }
10 }
```

Repare que na linha 1, o arquivo de implementação `leitura.c` incluiu o seu arquivo cabeçalho correspondente. Além disso, das linhas 3 a 9, houve a implementação da função `le_vetor`. Destaca-se que foi necessário a inclusão do arquivo `stdio.h`, uma vez que precisamos na função `printf`.

Uma coisa que você deve estar se perguntando é por que o nome do arquivo cabeçalho está entre aspas duplas e não entre os símbolos de < e > (como em `#include <stdio.h>`). A razão é simples, o cabeçalho `leitura.h` não está presente nos diretórios do sistema em que o compilador procura os arquivos cabeçalhos da biblioteca padrão do `C`, logo, o caminho relativo dele foi indicado entre aspas, para que o compilador o encontre.

Analogamente, podemos escrever o módulo de escrita de vetores. O arquivo cabeçalho deste módulo está disposto abaixo.

```
#ifndef ESCRITA_H
#define ESCRITA_H

void escreve_vetor(int vetor[],int n);

#endif
```

Por sua vez, o arquivo de implementação se encontra a seguir.

```
#include <stdio.h>
#include "escrita.h"

void escreve_vetor(int vetor[], int n){
        int i;
        for(i=0;i<n;i++){
                printf("V[%d] = %d\n",i,vetor[i]);
        }
}
```

Por fim, temos o módulo de operação entre vetores. Este módulo implementará a soma entre vetores, a subtração entre vetores e o produto escalar. O arquivo cabeçalho está abaixo.

```
#ifndef OPERACAO_H
#define OPERACAO_H


void soma_vetores(int vetor_1[],int vetor_2[],int vetor_resultado[], int n);
void subtrai_vetores(int vetor_1[],int vetor_2[],int vetor_resultado[], int n);
int produto_escalar(int vetor_1[],int vetor_2[], int n);

#endif
```

O arquivo implementação por sua vez, encontra-se disposto a seguir.

```
#include "operacao.h"

```

```c
void soma_vetores(int vetor_1[],int vetor_2[],int vetor_resultado[], int n){
        int i=0;
        for(i=0;i<n;i++){
                vetor_resultado[i] = vetor_1[i] + vetor_2[i];
        }
}

void subtrai_vetores(int vetor_1[],int vetor_2[],int vetor_resultado[], int n){
        int i=0;
        for(i=0;i<n;i++){
                vetor_resultado[i] = vetor_1[i] - vetor_2[i];
        }
}

int produto_escalar(int vetor_1[],int vetor_2[], int n){
        int i=0;
        int soma=0;
        for(i=0;i<n;i++){
                soma += vetor_1[i] * vetor_2[i];
        }
        return soma;
}
```

Os módulos foram implementados, mas como utilizá-los? Repare que nenhum deles possui a função `main`.

3 Utilizando os Módulos

Vamos criar um arquivo chamado `main.c`, que possui a função `main` e inclui os três arquivos cabeçalhos criado. Este arquivo lê dois vetores e computa a soma dos vetores, a subtração dos vetores e o produto escalar dos vetores. Após isso, ele imprime os vetores computados e o produto escalar. O código segue abaixo.

```c
#include <stdio.h>
#include "leitura.h"
#include "escrita.h"
#include "operacao.h"

#define MAX 100

int main(void){
        int vetor_1[MAX];
        int vetor_2[MAX];
```

```
11          int vetor_soma[MAX];
12          int vetor_subtracao[MAX];
13          int prod_escalar;
14          int n;
15          printf("Digite o tamanho dos vetores (<=100): ");
16          scanf("%d",&n);
17
18          printf("Leitura do vetor 1.\n");
19          le_vetor(vetor_1,n);
20          printf("Leitura do vetor 2.\n");
21          le_vetor(vetor_2,n);
22
23          /* Calcula soma de vetores */
24          soma_vetores(vetor_1,vetor_2,vetor_soma,n);
25
26          /* Calcula subtração de vetores*/
27          subtrai_vetores(vetor_1,vetor_2,vetor_subtracao,n);
28
29          /* Calcula o produto escalar*/
30          prod_escalar = produto_escalar(vetor_1,vetor_2,n);
31
32          printf("Imprimindo resultado da soma dos vetores.\n");
33          escreve_vetor(vetor_soma,n);
34
35          printf("Imprimindo resultado da subtração dos vetores.\n");
36          escreve_vetor(vetor_subtracao,n);
37
38          printf("Produto escalar = %d.\n",prod_escalar);
39
40          return 0;
41
42  }
```

3.1 Compilando Tudo

Para compilar todos os arquivos em um único executável `main` basta executar o comando `gcc escrita.c leitura.c operacao.c main.c -Wall -o main`.

3.2 Criando uma biblioteca

É possível utilizar os módulos produzidos para criar uma biblioteca que lida com vetores. Para isto é necessário compilar todos os arquivos de implementação em seu código objeto, isto é: `gcc -Wall -c leitura.c escrita.c operacao.c`.

Em seguida, arquive todos os códigos objeto em uma único arquivo: `ar -rc vetores.a leitura.o escrita.o operaco.o`. Sua biblioteca estática chamada `vetores.a` foi criada. Caso queira repassar o seu código para uma segunda pessoa sem expor a implementação, basta enviar o arquivo `.a` junto com os arquivos cabeçalhos. Por fim, esta pessoa deverá apenas *linkar* o código dela com a sua biblioteca, da seguinte forma:

`gcc main-colega.c -Wall vetores.a -o main-colega`.