Universidade Federal do Rio Grande do Norte
Departamento de Informática e Matemática Aplicada
Linguagem de Programação I • DIM0120

◁ Introdução à Linguagem C++, parte 1 (autor: Marcelo Siqueira) ▷

13 de fevereiro de 2019

1 O Ambiente de Programação

Um ambiente de programação na linguagem C++ consiste tipicamente de cinco componentes:

- editor,
- pré-processador,
- compilador,
- *linker* e
- *loader*

como mostrado na Figura 1.

O programador utiliza o **editor** para criar e/ou modificar seu programa e, depois, guardá-lo em disco. Nesta disciplina, você pode utilizar o editor de sua escolha, como por exemplo `vim`, `sublime`, GNU `Emacs` do Projeto GNU.

Para programar em C++, você necessita entender os passos e ferramentas do processo de compilação. O primeiro passo deste processo consiste na execução de um **pré-processador** no código-fonte que você criou. O pré-processador é um programa simples que substitui certos padrões no código-fonte por outros que você definiu usando *diretivas de pré-processamento*. Tais diretivas são usadas para "poupar" digitação e para aumentar a legibilidade do código. Por exemplo,

```
#define NUM 10
```

é uma diretiva que indica que toda ocorrência de `NUM` em seu programa-fonte deve ser substituída por `10`. O pré-processador também realiza outros tipos de substituiçao que veremos mais adiante.

Um código-fonte na linguagem C++ pode consistir de um ou mais arquivos com extensão `.cpp` (ou `.cc`[1]) e `.h` (ou `.hpp`[2]). Um arquivo com extensão `.cpp` contém, em geral, código para os métodos das classes e das funções definidas e declaradas, respectivamente, em arquivos com extensão `.h`.

O segundo passo é a geração de código de máquina propriamente dita. O módulo **compilador** recebe como entrada o código-fonte pré-processado, que comumente está guardado em um arquivo temporário gerado pelo pré-processsador. A saída gerada pelo compilador é o que denominamos de *arquivo objeto*. Um arquivo objeto contém código em linguagem de máquina correspondente ao código-fonte que você escreveu. Em ambientes de programação baseados no sistema operacional

[1] Esta extensão é comumente utilizada por programadores que usam sistemas operacionais baseados no Unix.
[2] O uso desta extensão é muito raro.

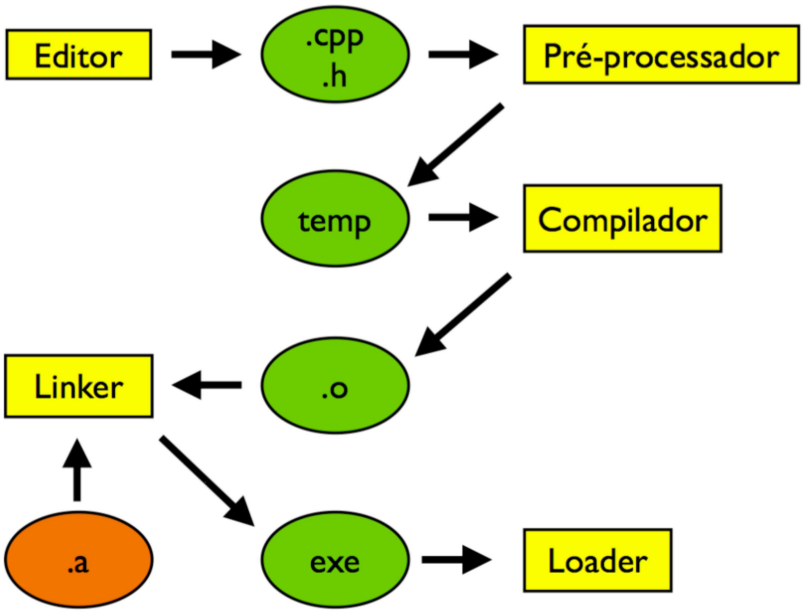

Figura 1: Elementos de um ambiente de programação na linguagem C++.

`Unix`, um arquivo objeto usualmente possui extensão `.o`. O compilador gera um arquivo objeto para cada arquivo do código-fonte que possui extensão a `.cpp`.

Apesar de estar escrito em linguagem de máquina, um arquivo objeto não é um programa executável. Isto porque o código-fonte pode ser composto de vários arquivos com extensão `.cpp`. Neste caso, um arquivo objeto foi gerado pelo compilador para cada arquivo `.cpp`. Um arquivo objeto faz referência a elementos de código que estão em outros arquivos objetos. Logo, esses arquivos precisam ser unificados em um único arquivo, o *arquivo executável*, contendo todas as instruções de máquina correspondentes ao código-fonte. Este papel de unificação de arquivos cabe ao **linker** e ele corresponde ao terceiro passo do processo de compilação do código-fonte.

Mesmo que o código-fonte consista de apenas um arquivo com extensão `.cpp` (e um ou mais arquivos com extensão `.h`), ele provavelmente utiliza funções cujos códigos de máquina estão em outros arquivos objetos. Por exemplo, *classes* ou funções que nos permitam ler e escrever dados, tais como `cin`, `cout`, `scanf` e `printf`. O código de máquina dessas funções estão organizados em arquivos denominados de *biblioteca*. Em ambientes baseados no `Unix`, um arquivo de biblioteca possui tipicamente extensão `.a`. O código de máquina de todas as classes e funções utilizadas pelo código-fonte também precisa ser anexado ao arquivo objeto do código-fonte.

Finalmente, um arquivo executável é executado com o auxílio do **loader**, que é um módulo do sistema operacional responsável por levar o arquivo executável armazenado em disco para a memória principal do computador. Quando o executável é posto em execução, ele se torna um *processo*.

2 Compilação com o GNU g++

Para ilustrar o processo de compilação, vamos utilizar o o editor de texto para criar o arquivo `lb01-01.cpp` abaixo *sem incluir os números das linhas*:

```
 1 //
 2 // Programa: lb01-01.cpp
 3 //
 4
 5 #include <iostream>     // cout, endl
 6
 7 // funcao principal
 8 int main()
 9 {
10   //
11   // Imprime uma mensagem no dispositivo de saida padrao.
12   //
13   std::cout << "Bem-vindo a linguagem C++!" << std::endl;
14
15   //
16   // Indica que o programa terminou com sucesso.
17   //
18   return 0;
19 }
20 // fim da funcao principal
```

A linha 5

`#include <iostream>`

é uma diretiva que solicita ao pré-processador para incluir o arquivo de cabeçalho `iostream.h` em seu código-fonte. Este arquivo contém as definições das classes que gerenciam entrada e saída de dados na linguagem C++. Nós precisamos incluir este arquivo para que o compilador possa reconhecer o objeto `cout` e a constante `endl`, que estão definidos em `iostream.h`.

Você pode verificar o resultado do pré-processamento do seu código-fonte executando a seguinte linha de comando:

`g++ -E lb01-01.cpp -o lb01-01-pp.cpp`

O que fizemos acima foi utilizar o compilador GNU g++ com a opção `E`, que solicita ao compilador para pré-processar o código apenas. Se você abrir o arquivo `lb01-01-pp.cpp` usando o `Emacs`, verá que seu código-fonte está no final do arquivo e vem seguido pelo conteúdo de `iostream.h`.

Você também pode compilar o arquivo `lb01-01.cpp` para gerar seu código objeto apenas:

`g++ -c lb01-01.cpp`

O resultado da compilação acima é um arquivo objeto denominado `lb01-01.o`. Verifique a existência deste arquivo objeto com o comando `ls`. Para gerar o arquivo objeto apenas, utilizamos a opção `c` do compilador.

Obviamente, podemos gerar um arquivo executável com uma única execução do compilador:

`g++ lb01-01.cpp -o lb01-01`

Com a linha de comando acima, o compilador produz como saída apenas o arquivo executável `lb01-01`.

Vamos considerar um código-fonte com mais de um arquivo com extensão `.cpp`. Quando isto acontece, apenas um dos arquivos contém a função `main` e os demais contêm classes e/ou funções que são referenciadas de dentro da função `main` ou por outras classes e funções no mesmo arquivo ou em arquivos distintos. No exemplo abaixo, temos três arquivos: `lb01-02a.cpp`, `lb01-02b.cpp` e `lb01-02a.h`. A função `main` está no primeiro deles apenas.

```
1 //
2 // Programa: lb01-02a.cpp
3 //
4
5 #include "lb01-02a.h"     // mensagem
6
7 // Função principal
8 int main()
9 {
10   //
11   // Imprime uma mensagem no dispositivo de saída padrão.
12   //
13   mensagem();
14
15   //
16   // Indica que o programa terminou com sucesso.
17   //
18   return 0;
19 }
20 // Fim da função principal
```

```
1 //
2 // Programa: lb01-02b.cpp
3 //
4
5 #include <iostream>     // cout, endl
6
7 //
8 // Definição da função mensagem
9 //
10 void mensagem()
11 {
12   std::cout << "Bem-vindo à linguagem C++!" << std::endl;
13
14   return;
15 }
16 // Fim da função mensagem
```

```
1 //
2 // Programa: lb01-02a.h
3 //
4
5 void mensagem();    // Declaração de função
```

Podemos compilar os dois arquivos com extensão `.cpp` separadamente e em qualquer ordem:

`g++ -c lb01-02a.cpp`

e

`g++ -c lb01-02b.cpp`

Use o comando `ls` para se certificar que os dois arquivos objetos foram criados.

Agora, podemos gerar um arquivo executável, digamos `lb01-02`, a partir dos dois arquivos objetos, `lb01-02a.o` e `lb01-02b.o`:

```
g++ lb01-02a.o lb01-02b.o -o lb01-02
```

Alternativamente, podemos usar apenas uma execução do compilador g++ para gerar o arquivo executável a partir dos dois arquivos com extensão `.cpp`:

```
g++ lb01-02a.cpp lb01-02b.cpp -o lb01-02
```

No caso acima, o compilador produzirá como saída apenas o arquivo executável; isto é, os dois arquivos objetos serão gerados como arquivos intermediários e, em seguida, serão apagados.

A linha 5 do arquivo `lb01-02a.cpp`

```
#include "lb01-02a.h"
```

solicita ao pré-processador para incluir o arquivo de cabeçalho `lb01-02a.h` no código em `lb01-02a.cpp`. Isto porque a função `main` em `lb01-02a.cpp` faz referência à função `mensagem`, que está *definida* em um outro arquivo `.cpp`: `lb01-02b.cpp`. Quando o compilador gera código objeto para `lb01-02a.cpp`, ele precisa saber o que a palavra `mensagem` significa. A inclusão do arquivo `lb01-02a.h` faz com que a *declaração* da função `mensagem` seja anexada, pelo pré-processador, ao código em `lb01-02a.cpp`. Desta forma, quando o compilador for gerar código para `lb01-02a.cpp`, ele já saberá o que a palavra `mensagem` significa. Note que, neste ponto, o compilador não precisa conhecer o código de `mensagem`, que está no arquivo `lb01-02b.cpp`.

A linha 5 do arquivo `lb01-02b.cpp`

```
#include <iostream>
```

solicita ao pré-processador para incluir o arquivo de cabeçalho `iostream.h` no código em `lb01-02b.cpp`. Isto porque a função `mensagem` em `lb01-02b.cpp` faz referência a `cout` e `endl`.

◀ FIM ▶