# Começando

Gonzalo Travieso

2019

## 1 Ponto de entrada

A execução de um programa C++ começa[1] em uma *função* denominada `main`. Mais tarde veremos os detalhes sobre funções, agora mostramos o seu uso para começar o programa, através da apresentação do que seria o mais simples programa possível:

```
int main(int, char*[])
{

}
```

Este programa não faz nada, terminando logo depois de começar. Mas vejamos as suas partes:

- `main` é o nome da função "principal" de um programa C++, como já dito, aquela que começa executando ao se rodar o programa.
- O conteúdo entre parêntesis é a *lista de parâmetros* da função. Não discutiremos os parâmetros aqui, pois não serão utilizados.
- `int` é o valor de retorno da função, e indica um código inteiro que será retornado pelo programa ao sistema operacional, quando terminar a execução. O uso desse valor depende do sistema operacional, mas a convenção é que o retorno do valor `0` indica que o programa acabou sem erros, enquanto um valor diferente deste indica algum tipo de erro. Se não indicamos explicitamente o valor a retornar, o C++ retorna automaticamente um `0`.
- Entre o `{` e o `}` são colocados os comandos a serem executados pelo programa. Como não há nada aqui, este programa não executa nenhum comando, terminando logo depois de ser iniciado.

## 2 Compilação e execução

Para executar esse (muito útil) código, é necessário transformá-lo em algo que o computador entenda, com uma ajuda dos sistema operacional. Precisamo inicialmente colocar esse código em um arquivo, tradicionalmente com a extensão

[1] Veremos mais tarde que existem exeções.

`.cpp`; esse arquivo é denominado o *código fonte* do programa. Vamos supor que esse programa esteja guardado em um arquivo com o nome `inutil.cpp`. O computador não sabe como executar esse arquivo, pois ele é apenas uma sequência de caracteres. Para poder executá-lo, precisamos transformar o programa com a ajuda de um *compilador*[2]. O compilador irá analisar o código fonte e a partir dele gerar o *código executável* em um novo arquivo. Na tradição UNIX, o código executável tem o mesmo nome do código fonte, mas **sem extensão**.

Aqui vamos usar dois compiladores: o compilador GNU e o CLANG. Com o compilador GNU, podemos gerar o código executável para o nosso programa da seguinte forma:

```
g++ -std=c++17 -Wall -Wextra -Wpedantic -O2 inutil.cpp -o inutil
```

O significado das diversas partes desse comando é o seguinte:

- `g++` é o programa da coleção de compiladores GNU responsável por compilar códigos C++.
- `-std=c++17` indica que estamos usando o padrão C++ definido em 2017 (o mais recente). Se não especificamos isso, o compilador assume algum que depende da versão do compilador.
- `-Wall` indica que estamos pedindo para o compilador dar avisos sobre possíveis problemas no código (mesmo que esses não sejam erros de acordo com a linguagem).
- `-Wextra` pede para incluir alguns avisos que não são incluídos em `-Wall`.
- `-Wpedantic` verifica se o código está estritamente correto (do ponto de vista da linguagem, e não de correção de acordo com a intenção do programador) de acordo com o padrão.
- `-O2` pede para ser feita extensiva otimização de código.
- `inutil.cpp` é o nome do arquivo com o código fonte que queremos compilar.
- `-o` indica que o nome que segue (neste caso `inutil`) deve ser o nome do arquivo com o código executável.

Para usar o compilador CLANG, basta substituir o `g++` por `clang++`:

```
clang++ -std=c++17 -Wall -Wextra -Wpedantic -O2 inutil.cpp -o inutil
```

Neste programa simples, a inclusão de todas essas opções é desnecessária, mas já quero que vocês se acostumem a incluir essas opções nas suas compilações.

Após essa compilação, podemos executar o código:

```
./inutil
```

O `./` antes do nome do arquivo indica que o arquivo a executar deve ser procurado no diretório corrente.

[2] Na verdade, existem várias etapas nesse processo, e o uso do termo *compilador* para designar todas elas é inacurado, mas comum.

# 3 Saudação

Vejamos agora um código que faz alguma coisa: o famoso e tradicional "Hello, world!".

```
/*
 * Acho que voces ja conhecem uma variante do codigo abaixo.
 */
#include <iostream>

// Saudacao a todos
int main(int, char *[])
{
    // Nada de original.
    std::cout << "Hello, world!\n";
}
```

Neste código temos alguns elementos adicionais:

- O `std::cout` é um objeto (veremos os detalhes mais tarde) que permite escrever coisas no terminal onde o programa está sendo executado.[3] Para isso basta usar o operador `«`, como acima.
- O `std::cout` usado precisa ser definido em algum lugar. Isto é feito em um arquivo denominado `iostream`, que faz parte da biblioteca de C++. Esse arquivo deve ser incluido (usando `#include`) no programa para podermos usar o `std::cout`.
- O ponto-e-vírgula no final da linha da mensagem indica o final de um comando. Todos os comandos em C++ devem ser terminados por um ponto-e-vírgula.
- Comentário em C++ podem ser indicados de duas formas:
  - O caso normal é usarmos `//`, caso em que tudo o que segue até o final da linha é considerado um comentário.
  - O outro caso é quando queremos marcar um conjunto de linhas consecutivas como comentários. Neste caso, colocamos `/*` no começo do comentário e `*/` no final. Note que qualquer `*/` que aparecer termina o comentário, e portanto `/*` e `*/` não podem ser aninhados:

    ```
    // ISTO ABAIXO É UM ERRO:
    /* Comentando /*um comentario*/ nao da certo. */
    ```

[3] Na verdade ele escreve na *saída padrão* do sistema, que costuma ser o terminal que executou o programa, mas pode ser outra coisa.