Aula 1: Tipos básicos e namespace

Kevin Wallacy de Souza Maciel

4 de outubro de 2018

Introdução

Esta aula tem como objetivo introduzir os tipos básicos da linguagem c++ (int, float, bool, char) e falar brevemente sobre namespaces e sua funcionalidade.

1 Tipos Básicos

Na linguagem C++ existem 5 tipos básicos: *int, float, double, bool, char*.

O que significam esses tipos?

Tipos de dados são rótulos para que o compilador saiba interpretar e alocar corretamente o tamanho daquela variável (como por exemplo, um tipo int utiliza 4 bytes de memória e um char utiliza apenas um byte).

1.1 Inteiro

O tipo int representa um inteiro de –2,147,483,648 até 2,147,483,647, que ocupa 4 bytes de memória.

Este tipo pode vir seguido de modificadores, como: unsigned, long, short. Estes modificadores podem alterar o tamanho em bytes (como o long ou short) ou o intervalo de representação (como o unsigned que muda o intervalo para [0, 4,294,967,295]).

Também é possível realizar operações básicas como adição, subtração, multiplicação e divisão utilizando os operadoes +, -, * e / respectivamente.

Existem também operadores de comparação, como == (igualdade), != (diferença), < (menor que), > (maior que), <= (menor igual que), >= (maior igual que), o operador de atribuição (que atribui um valor a uma variável) representado por = e o operador de resto de divisão (retorna o resto da divisão inteira entre dois números) representado por %.

1.2 Pontos flutuantes

Os tipos float e double representam números reais (ponto flutuante), o tipo float representa um real com até 7 dígitos decimais e o double até 15 dígitos decimais.

Eles ocupam respectivamente, 4 e 8 bytes de memória.

Similar ao int, eles também podem vir seguidos de modificadores para alterar seu tamanho ou intervalo numérico.

Também similar aos inteiros, estes tipos também podem realizar operações básicas como adição, subtração, multiplicação, divisão, comparação e atribuição utilizando os mesmos operadores.

1.3 Char

O tipo char representa caracteres como **a, b, c, d, e, f, !, =, /, -,** | e afins.

Este tipo ocupa 1 byte de memória e também pode vir acompanhado de modificadores como o unsigned.

O tipo char não possui operadores aritiméticos (+, -, *, / e %), mas pode utilizar os operadores de comparação e atribuição assim como os inteiros.

1.4 Booleano

O tipo booleano representa os valores booleanos **true** (1) e **false** (0).

Este tipo de dado utiliza 1 byte de memória e não possui modificadores como short ou unsigned diferente dos tipos que vimos enteriormente.

Este tipo, similar ao char, não possui operadores aritiméticos mas pode utilizar os operadores de comparação e atribuição.

2 Tipos especiais

Fora estes tipos básicos, também existem certos tipos especiais com o std::string, std::vector e vários outros tipos que podem ser encontrados em sites ou bibliotecas personalizadas.

Estes tipos especiais geralmente são implementados por classes, que são **objetos** criados pelo programador para atender certas necessidades da aplicação, vamos aprender mais sobre elas em aulas futuras.

3 Namespace

Como foi citado anteriormente, existem tipos especiais como o std::string por exemplo, mas o que significa este **std::** antes do nome do tipo?

O prefixo **std::** representa um namespace que serve como uma espécie de rótulo ou sobrenome daquele tipo (neste caso, o tipo string).

Dessa forma, ao chamar **std::string** estamos especificando que queremos utilizar o tipo string que foi definido no namespace **std**.

É como se numa turma existissem dois alunos com nomes iguais, ambos chamados joão por exemplo. Caso o professor precise falar diretamente com um deles, ele terá que usar um sobrenome pra não causar confusão (ex: um se chama joão maurício e o outro joão vidal).

4 Impressão

Uma vez que sabemos quais são os tipos, seus operadores, e o que é um namespace, vamos aprender como interagir com estes tipos via impressão e coleta de valores pelo

terminal.

Com impressão de valores, nos remetemos à ação de mostrar na tela uma mensagem ou o dado guardado por uma variável para podermos entender o que se passa dentro do nosso código.

Já com relação a coleta, nos remetemos à ação de coletar dados via terminal para que com estes dados, consigamos realizar certas funcionalidades do algorítimo.

Interessante não? vamos ver como realizar cada um com mais detalhes.

4.1 Métodos de impressão

Para realizar a impressão de valores, precisamos utilizar o seguinte comando: **std::cout«"Texto";**. Vamos dissecar este comando:

4.1.1 cout

O comando **cout** representa um objeto da classe **std::ostream**. Este objeto é responsável em estabelecer um canal de comunicação entre o algorítimo e o terminal (objeto de saída) e a partir deste canal, podemos inserir os dados a serem impressos na tela.

4.1.2 Operador «

O comando « representa o operador de inserção do objeto **cout**, este operador é responsável por inserir os dados no canal representado por **cout**.

4.1.3 Texto

O comando **"Texto"** representa o dado a ser inserido na stream. Neste caso, o dado é um texto (delimitado por), mas também poderíamos utilizar o valor de uma variável. Basta substituir "Texto"por sua variável (Ex: std::cout«a; sendo a uma variável inteira, por exemplo).

4.2 Métodos de coleta

Para coletar dados via terminal, utilizamos o comando: **std::cin»a;**. Vamos dissecá-lo também.

4.2.1 cin

O comando **cin** representa um objeto da classe **std::istream** que similar ao **cout** estabelece um canal de comunicação entre o terminal e o algorítimo.

4.2.2 Operador »

O comando » representa o operador de descarga, que descarrega os dados de uma stream no operando de destino (normalmente uma variável).

4.2.3 a

O comando **a** representa uma variável genérica (pode ser um char, int, float e etc) que neste caso recebe a informação retirada da stream.

5 Compilação

Aprendemos os tipos, operadores, como exibir os dados mas ainda resta uma barreira: Como transformar isso num executável?

Ao concluir a montagem do código, o proximo passo para gerar o seu algorítimo é compilar o código.

Este curso irá abordar a compilação em sistemas linux, caso esteja utilizando outro sistema operacional, não garantimos que irá funcionar da mesma forma.

Para compilar seu código, basta (por meio do terminal) acessar o diretório onde se encontra o código e digitar o seguinte comando: **g++ -Wall -std=c++11 nome_do_arquivo.cpp -o nome_do_executável_desejado**.

Vamos entender melhor este comando.

5.1 g++

O comando **g++** representa o compilador padrão do C++ em ambientes linux.

5.2 -Wall

O comando **-Wall** representa uma flag de compilação, tais flags são parâmetros que passamos para o compilador (nesse caso o g++) para que ele execute ou não certas otimizações ou marcações (veremos mais sobre flags de compilação futuramente).

Mais especificamente, a flag **-Wall** representa que queremos receber os avisos de erros que possam vir a aparecer na compilação.

5.3 -std=c++11

O comando **-std=c++11** também é uma flag assim como o -Wall, mas neste caso, ela representa a versão do C++ em que desejamos basear nossa compilação (neste caso, a versão C++11).

5.4 nome_do_arquivo.cpp

Este comando representa o arquivo a ser compilado, basta digitar o nome do documento e sua extensão, caso a extensão seja válida, o arquivo será compilado.

5.5 -o

O comando **-o** representa uma flag de compilação que representa que desejamos nomear o arquivo executável resultante da compilação.

5.6 nome_do_executável_desejado

Este comando representa o nome que se deseja dar ao arquivo executável gerado após a compilação (caso não seja fornecido, o executável é nomeado a.out).

6 Conclusão

Uma vez que já vimos os tipos de dados, operadores, métodos de impressão e coleta de dados e como compilar o código vamos ver como aplicar este conhecimento (Abra o arquivo **Aula-1.cpp**).