Variáveis e tipos

Gonzalo Travieso

2019

1 Preliminares

Vimos que os dados em processamento são armazenados pelos computadores na memória como um conjunto de bytes que representam apenas valores binários.

Para dar um significado a esses valores binários precisamos saber o que eles estão representando; por exemplo, um conjunto de 4 bytes (32 bits) pode representar tanto um inteiro em complemento de dois quanto um número de ponto flutuante 32 bits padrão IEEE, ou 4 caracteres ASCII, dentre diversas outras possibilidades.

Assim, quando vamos utilizar um dado em um processamento, precisamos de duas informações sobre ele:

- Onde ele se encontra na memória; e
- Que tipo de dados é usado para sua representação.

Isto é feito em C++ através das **variáveis**, que representam uma localização específica na memória com um tipo de dados associado.

2 Declaração de variáveis

Quando usamos uma variável o *compilador*, que é o programa responsável pela tradução do nosso *código-fonte* em *código executável*, se encarrega de associar uma localização na mémória (endereço e número de bytes utilizados) com um *nome* ou *identificador* da variável. Assim, quando usamos o nome da variável, estamos acessando a posição de memória correspondente.

Para que isso ocorra, antes de utilizarmos uma variável precisamos fazer a sua **declaração**, indicando seu *identificador* e seu *tipo*. Isto é feito como no exemplo abaixo, que declara uma variável denominada `idade` do tipo `int`

```
int idade;
```

Em primeiro lugar, damos o nome do tipo (`int`, que corresponde a um inteiro em complemento de 2), depois o nome que desejamos para a variável (`idade`); por fim, terminamos com um `;`.

É possível declarar mais do que uma variável do mesmo tipo em um mesmo comando, usando vírgulas para separar os nomes das variáveis, mas isso não é prática recomendada.

3 Alguns tipos de C++

Em C++, alguns dos tipos são precisamente defindos pela linguagem, outros têm alguns detalhes de implementação deixados a cargo do compilador. Usaremos as definições para o caso dos compiladores GNU e CLANG, para máquinas de 64 bits (as mais comuns atualmente).

Alguns dos tipos mais usados em C++ são:

int representa um inteiro em complemento de 2 de 32 bits.

unsigned int representa um inteiro maior ou igual a 0.

char representa um caracter (8 bits) em representação ASCII.

float representa um número de ponto flutuante usando a representação IEEE-754 de 32 bits.

double IEEE-754 de 64 bits.

bool Representa uma variável booleana, que pode ser verdadeira (`true`) ou falsa (`false`).

Exemplos:

```
int a;
char c;
float x;
double y;
```

4 Constantes literais

Nem todos os valores usados em um programa estão armazenados em variáveis. Alguns deles aparecem **explicitamente** no código-fonte. Chamamos esses valores de *constantes literais*, pois eles aparecem escritos *literalmente* no código.

Como todos os valores em um programa, as constantes literais também têm um tipo; esse tipo é determinado pela forma como elas são escritas. Para os tipos citados acima:

int A constante pode ser fornecida em decimal, hexadecimal, octal ou binário. Para decimal, simplesmente escrevemos o número inteiro desejado, com sinal se necessário, mas sem usar ponto decimal e **sem colocar zeros no começo**. Para hexadecimal, precedemos o valor pelos caracteres `0x` ou `0X` e seguimos com os dígitos hexadecimais, usando os caracters `a` até `f` (minúsculo ou maiúsculo) para os dígitos de 10 até 15, respectivamente. Para octal, colocamos os dígitos octais de 0 a 7, **precedidos por um zero** (essa é a razão porque não podemos colocar zeros à esquerda quando usamos representação decimal). Para representação binária, precedemos o número com `0b` ou `0B`. Constantes literais inteiras com muitos dígitos podem ser tornadas mais claras com o uso de `'` como um separador de dígitos. Exemplos:

- `10` inteiro com valor 10.
- `10000000` inteiro com valor 10 milhões.

- `0x10` inteiro com valor 16 (notação hexadecimal).
- `0x10000000` inteiro com valor 268.435.456 (notação hexadecimal).
- `10'000'000` inteiro com valor 10 milhões.
- `0x10'00'00'00` inteiro com valor 268.435.456 (notação hexadecimal).
- `010` inteiro com valor 8 (notação octal).
- `010000000` inteiro com valor 2.097.152 (notação octal).
- `0b10` inteiro com valor 2 (notação binária).
- `0b10000000` inteiro com valor 128 (notação binária).

  Um detalhe técnico: as constantes especificadas em hexadecimal, octal ou binário podem ser consideradas `unsigned int` se não couberem em um `int` com valor positivo. Existem outras regras com relação a isso que não vamos abordar.

char Um caracter é (normalmente) representado por ele mesmo dentro de um par de '. Exemplo '`a`', '`Z`'. Alguns caracteres sem representação direta, como o caracter de mudança de linha, são representado por uma barra reversa seguida de um código especial. No caso da mudança de linha, usamos '`\n`' (este é apenas um caracter).

double Literais de ponto flutuante de precisão dupla são números que possuem pelo menos um dos seguintes: um ponto decimal ou uma potência de 10. Exemplos:

- `1.0` vale 1 em ponto flutuante de 64 bits.
- `1.` o mesmo que o anterior.
- `.5` este é o mesmo que `0.5`
- `1.2e2` vale o mesmo que `120.0`.
- `-1.2e-2` vale o mesmo que `-0.012`.
- `1e2` vale o mesmo que `100.0`

float São representados da mesma forma que os `double`, mas acrescentando um `f` ou `F` no final: `1.0f`, `1.F`, `-1.2e-2f`.

5 Inicialização

Quando declaramos variáveis como mostrado acima, não especificamos um valor para elas. Normalmente isso resulta em que elas terão valores não especificados (em geral ficam com o que estava no local de memória onde o compilador as colocou, isto é, lixo do ponto de vista do programa). Se queremos evitar essa situação (o que em geral é o caso), podemos declarar o valor inicial da variável quando a declaramos. Isso pode ser feito de diversas formas, mas as duas mais usadas são usando um `=` ou um par `{}` na declaração, como no exemplo abaixo.

```
int a; // Variavel a criada, mas nao sabemos seu valor
int b = 1; // Variavel b criada, com valor inicial 1
double c{0.0}; // Variavel c criada com valor inicial 0
```

O uso de inicialização é recomendado, principalmente considerando a característica de C++ de que podemos declarar uma variável em qualquer parte do código (e não apenas no começo do bloco, como em C). Assim, o normal é declararmos a variável antes de seu uso mas apenas após conseguirmos calcular seu valor inicial.

6 Conversão automática de tipos

Para simplificar o código, C++ permite que se usem valores de um tipo em locais onde se esperam valores de outro tipo. Neste caso, o tipo do valor fornecido é convertido implicitamente (automaticamente) para o tipo esperado, desde que o compilador conheça uma forma de realizar a conversão.

Um dos locais onde isso acontece é na inicialização de variáveis:

```
double soma = 0;
```

No exemplo acima, a variável é declarada como de ponto flutuante, mas a constante literal usada em sua inicialização é inteira. Como o compilador está esperando um valor de ponto flutuante, o 0 inteiro é convertido para esse tipo antes de ser usado na inicialização. Neste caso específico, a conversão é realizada pelo próprio compilador.

7 Dedução automática de tipo

Quando realizamos uma declaração de variáveis da forma

```
double soma{0.0};
```

O fato de estarmos lidando com `double` é expresso duplamente: primeiro, na especificação do nome do tipo e segundo no tipo da constante literal que fornece o valor inicial. Essa redundância pode ser eliminada (o que tem o efeito colateral de ajudar em futuras possíveis mudanças no código) utilizando um outro formato de declaração de variáveis:

```
auto soma{0.0};
```

Nesta situação, a palavra-chave `auto` significa que o compilador deve deduzir o tipo da variável `soma` de acordo com o tipo do valor inicial fornecido.

```
auto p{0};
auto q{p+1};
```

No exemplo acima, a variável `p` é definida como do tipo `int`, por ter o valor `0` usado na inicialização; já a variável `q` é definida como `int` pois esse é o tipo resultante da soma de `p` (um `int`) com 1 (também um `int`).

8 Dois tipos importantes da biblioteca

Os tipos de dados apresentados anteriormente são alguns dos pré-definidos na linguagem. Em C++, alguns dos tipos mais usados não fazem parte da linguagem, mas são definidos na biblioteca. Dois deles são usados para representar cadeias de caracteres e arranjos lineares de valores de mesmo tipo (chamados *arrays* ou *vetores*).

8.1 Cadeias de caracteres

Para trabalhar com cadeias de caracteres, devemos incluir a biblioteca `string` e então usar o tipo `std::string`. As constantes literais de cadeia de caracteres são um conjunto de caracteres ASCII delimitados por `"` seguido de um `s`. O `s` final pode ser omitido em diversas situações.

```
#include <string>
// ...

std::string jo{"Joaquim"s}; // OK
std::string maria{"Maria"}; // OK
jo = "Joaquim Carlos"; // OK
auto ana{"Ana"s}; // OK, ana tem tipo std::string
auto pedro{"Pedro"}; // OOPS: Pedro tem tipo char const * (???)
```

8.2 Vetores

Existem diversas opções para representar arrays em C++, mas a de uso recomendado para a maioria das situações é o `std::vector`. Para usá-lo, precisamos incluir a biblioteca `vector` e declarar a variável com o tipo `std::vector<T>`, onde `T` deve ser substituido pelo tipo dos elementos armazenados no vetor.

```
#include <vector>

// ...

std::vector<int> vi(10); // vi tem 10 valores int
std::vector<double> vd(100); // vd tem 100 valores double
std::vector<int> alguns{1, 2, 4, 8}; // alguns tem os valores 1, 2, 4 e 8
```

Para acessar cada um dos componentes do vetor, usamos **indexação**, colocando o índice da posição do componente que queremos acessar entre `[` e `]`.

```
vi[0] = 1;
vi[1] = vi[0] + 3;
```

Se um vetor tem N elementos, os valores válidos para índices serão de 0 a $N-1$. **É responsabilidade do programador garantir que valores de índices inválidos não sejam usados!**