Variáveis e constantes

Declaração

Ao contrário de linguagens como Python, em C toda variável deve ser declarada antes de ser usada, e o tipo deve ser especificado na declaração. Exemplos:

```
int x;
double nota;
unsigned char c;
volatile unsigned long long int declaracaoLonga;
```

É possível atribuir um valor à variável já durante a declaração:

```
int x = 100;
double nota = -2.1;
unsigned char c = 'F';
```

Nota: nomes de variáveis não podem começar com dígitos, nem com underscore seguido de maiúscula. Não se pode usar dois underscores seguidos em nenhum lugar no nome da variável.

Tipos primitivos

- `int`: inteiro
- `float`: número real (ponto flutuante)
- `double`: número real (ponto flutuante, com mais precisão que o `float`)
- `char`: byte (onde cabe 1 caractere ASCII)
- `bool` (apenas nas versões mais novas de C): um valor lógico, que pode ser `true` ou `false`.

Nota: `char` não é sinônimo de caractere.
Na época em que C foi criado, a codificação mais usada era ASCII, e todo caractere ASCII cabe em 1 byte. Hoje, a codificação mais usada é UTF-8, na qual um caractere pode ocupar vários bytes; portanto, podem ser necessários vários `char`s para guardar um único caractere.

Nota: em C, existe uma correspondência direta entre `int` e `bool`: `0` é `false`; qualquer outro valor é `true`. (Geralmente usa-se `1` para `true`.) Em função disso, nas versões de C que não têm `bool`, usa-se simplesmente `int` no lugar.

Modificadores de tipo

- `long`: pode aumentar o tamanho do tipo. Usável com: `int` e `double`.
  (nota: em alguns casos, `long int` tem o mesmo tamanho que `int`).
- `long long`: aumenta o tamanho do tipo. Usável com: `int`.
  (nota: `long long int` tem no mínimo 8 bytes)
- `short`: pode diminuir o tamanho do tipo.
  (nota: em alguns casos, `short int` tem o mesmo tamanho que `int`).
- `signed` e `unsigned`: altera a definição do tipo para que seja considerada com sinal (inclui números negativos) ou sem sinal (apenas zero e números positivos). Usável com: `int` e `char`
- `const`: especifica que aquele nome não corresponde a uma variável, mas sim a uma constante. Qualquer tentativa de alterar o valor da constante gera um erro de compilação. Por convenção, usam-se nomes em maiúsculas para constantes, assim como para macros (e pelo mesmo motivo; veja #define abaixo). Usável com: qualquer tipo.

Exemplos:

```
const int BUFFER_SIZE = 4096;
const double PI =  3.1415926535;
```

Nota: quando se usa um desses modificiadores (à exceção do `const`) com `int` é possível omitir o `int`; por exemplo, `long int x;` pode ser escrito simplesmente como `long x;`.

Conversão de tipo (cast)

Em C, uma conversão de tipos é feita como:

```
(tipo_para_o_qual_converter) expressao
```

Por exemplo:

```
int x = 5;
float y = (float)x / 2;
```

Arrays

Uma array é uma sequência de valores de mesmo tipo, alocados em endereços contínuos na memória. Arrays são declarados com o número de elementos após o nome, entre colchetes `[]`.
Por exemplo:

```
int arr[20]; /* declara uma array de 20 ints */
```

Como qualquer outra variável, é possível atribuir um valor inicial para a array durante a declaração:

```
int arr[5] = {10, -1, 4, -7, 31}; /* todos os elementos são especificados */
int arr[20] = {10, -1, 4, -7, 31}; /* apenas até o 5o é especificado */
int arr[3] = {10, -1, 4, -7, 31}; /* ERRO! A array não é grande o suficiente */
```

Nesse caso, é possível também omitir o tamanho da array e deixar que o compilador determine o tamanho:

```
/* o compilador declara a array com o menor tamanho possivel que caiba todos os elementos, ou seja, 5 */
int arr[] = {10, -1, 4, -7, 31};
```

Usa-se o operador `[]` para acessar o n-ésimo elemento da array. Por exemplo:

```
int arr[20];
arr[0] = 1;
arr[1] = 2 + arr[0];
arr[2] = arr[1] - 2 * arr[0];
/* ... */
arr[19] = -10;
```

Se a array foi declarada com `N` elementos, é possível acessar os elementos de `0` a `N - 1`. Em C, ao contrário de lingaugens como Java, não há checagem se o indíce está dentro dos limites da array.

`arr[i]` é, por definição, equivalente a `*(arr + i)`; vide relação com arrays.

Strings

Em C, strings são sequências de caracteres terminadas pelo caractere especial `\0`. Strings são implementadas como ponteiros ou arrays de `char`. (Ponteiros serão explicados mais adiante.)

Literais

Literais (literals) são representações de um valor fixo. Os exemplos mais óbvios são números como `21` ou `3.14`, mas há também:

- char literals: um único caractere (ou backslash sequence; veja abaixo) entre aspas simples. Exemplos: `'a'`, `'J'`, `'\n'`, etc.
- string literals: um ou mais caracteres ou backslash sequences entre aspas duplas. Exemplos: `"foo"`, `"Hello World"`, etc.

Todo string literal, assim como toda string em C, é terminado pelo caractere especial `\0`.
É possível atribuir um char literal a uma variável do tipo `char`, e um string literal a um ponteiro ou array de `const char`:

```
const char *string_ponteiro = "foo";
const char string_array[] = "bar";
```

(Nota: é possível omitir o `const` nos dois casos, mas tentar alterar um caractere de uma string literal é undefined behaviour.)

Caracteres escapados (backslash sequences)

Esses caracteres podem ser usados tanto em char literals quanto em string literals:

- `\n`: newline
- `\t`: tab
- `\x<H1><H2>`: o byte de valor 0x<H1><H2>
- `\r`: carriage return (em arquivos de texto do Windows, o final da linha é representado por `\r\n`, equanto Linux usa apenas `\n`)
- `\b`: backspace (pode ser usado para sobrescrever algo que já foi impresso)

(Há outros, esses são os mais importantes.) Para obter uma barra invertida (backslash) de verdade, use `\\`.

Escopo

Em C, todo código que está entre chaves `{` e `}` é chamado de bloco. O escopo da variável é o bloco em que ela foi declarada; ou seja, a variável só existe dentro daquele bloco. Por exemplo:

```
int x;
if(1) {
  int y;
  y = 5; /* OK */
  x = 0; /* OK, declarada num escopo mais externo */
}
x = 1; /* OK */
y = 0; /* ERRO! Variável fora de escopo! */
```

É possível declarar variáveis com o mesmo nome em escopos diferentes. A variável usada é sempre a do escopo mais interno. Por exemplo:

```
int x = 0;
if(1) {
  int x;
  x = 1;
  printf("x = %d\n", x);
}
printf("x = %d\n", x);
```

Tem como output

```
x = 1
x = 0
```

Ou seja, a atribuição dentro do if não altera a variável fora do if.

Nota: nas versões mais antigas de C, as declarações de variáveis só podiam ser feitas no começo do bloco. Versões posteriores não têm essa restrição.

Revision #1
Created Mon, Jan 28, 2019 11:30 PM by Luana
Updated Mon, Jan 28, 2019 11:32 PM by Luana