Tipos de dados

Em C, as variáveis têm seu tipo pré-definido no código, ou, dizemos que seu tipo é *estático*. Em princípio, o tipo de uma variável não pode ser mudado, embora possa ser contornado através de operações especiais como o *type casting*, como veremos adiante.

Os principais tipos de dados em C são:

- Tipos escalares
  - Tipos aritméticos
  - Inteiros
  - Reais
  - Complexos (C99)
  - Tipos ponteiros
- Tipos não-escalares
  - Estruturas (structs)
  - Uniões
  - Arranjos (vetores e matrizes)

Tipos Inteiros

Os tipos inteiros básicos no padrão ANSI são mostrados abaixo. O "tamanho" do tipo (sizeof) é o número de bytes necessários para a representação.

Tipo	sizeof	Faixa de valores
char	1	-128...127 ou 0...255 (depende da implementação)
unsigned char	1	0...255
signed char	1	-128...127
int	2 ou 4	-32.768...32.767 (32 bits) -2.147.483.648...2.147.483.647 (64 bits)
unsigned int	2 ou 4	0...65.535 (32 bits) ou 0...4.294.967.295 (64 bits)
short	2	-32.768...32.767
unsigned short	2	0...65.535
long	4	-2.147.483.648...2.147.483.647
unsigned long	4	0...4.294.967.295

O arquivo "limits.h" define constantes para os limites de valores aceitos dos tipos inteiros, como "INT_MIN", "LONG_MAX", etc. No Linux, esse arquivo se encontra em "/usr/include/limits.h".

O padrão C99 definiu mais alguns tipos inteiros:

Tipo	Tamanho (bytes)	Faixa de valores
long long	8 bytes	-9.223.372.036.854.775.807...9.223.372.036.854.775.807
unsigned long long	8 bytes	0...18.446.744.073.709.551.615

Como alguns tipos inteiros básicos dependem da implementação do compilador ou da plataforma de execução, o padrão C99 trouxe definições de tipos inteiros **de tamanho fixo** (encontradas no arquivo "inttypes.h"), como "int8_t" (inteiro de 8 bits com sinal), "uint32_t" (inteiro de 32 bits sem sinal), etc. Mais informações podem ser encontradas nesta página.

Tipos Reais

Os tipos reais (de ponto flutuante) básicos são indicados abaixo.

Tipo	sizeof	Faixa de valores	Precisão
float	4	1.2E-38...3.4E+38	6 dígitos
double	8	2.3E-308...1.7E+308	15 dígitos
long double	10	3.4E-4932...1.1E+4932	19 dígitos

Tipo booleano

A especificação C ANSI original **não possui** um tipo booleano. Em seu lugar, toda variável ou expressão cujo resultado seja zero (ou nulo) é avaliada como FALSA; caso o resultado seja diferente de zero (ou não-nulo), a variável ou expressão é considerada VERDADEIRA, inclusive para valores negativos.

A especificação C99 acrescenta um tipo booleano (bool), acessível através do arquivo "stdbool.h".

O Tipo void

A palavra reservada "void" designa um valor ou variável de tipo desconhecido. Esse tipo é usado em algumas situações especiais que serão vistas posteriormente, como ponteiros para variáveis cujo tipo é desconhecido a priori, ou para indicar que uma função não retorna nenhum valor válido.

Constantes

O uso de constantes ajuda a documentar o código. Constantes são definidas com "#define's", e por convenção, constantes são escritas em maiúsculas.

C é *case sensitive* e portanto "achtung" é diferente de "Achtung" que é diferente de "ACHTUNG".

Seguem alguns exemplos de constantes do tipo inteiro.

```
#define GRANDE  1000
#define PEQUENO 10
#define MEDIO   ((PEQUENO + GRANDE) / 2)
```

Os caracteres de controle podem ser representados pelos seus *escapes*, como '\t' para a tabulação horizontal, ou por apelidos:

```
#define HTAB    '\t'      // tabulação horizontal
#define BKSP    '\b'      // backspace
#define NEWL    '\n'      // newline
#define BELL    '\007'    // bell, representado em octal (3 dígitos, 0..7)
#define VTAB    '\xb'     // tabulação vertical, em hexadecimal (prefixo x)
```

Um **tipo enumerado** pode ser usado para definir os caracteres de controle. Os códigos dos caracteres podem ser representados em hexadecimal, com prefixo "0x", ou pelas *sequências de escape* como '\n'.

```
enum ctrl_chars_x { BELL = 0x07 , BKSP = 0x08, HTAB = 0x09, NEWL = 0x0a }
enum ctrl_chars_e { BELL = '\a' , BKSP = '\b', HTAB = '\t', NEWL = '\n' }
```

O compilador reclamaria destas definições porque os nomes usados nas duas enumerações *devem* ser distintos, embora os valores possam ser repetidos.

Um tipo enumerado pode definir os 'números' que correspondem aos meses. Neste exemplo, o primeiro valor é definido e os seguintes são atribuídos pelo compilador, da maneira óbvia: FEV=2, MAR=3, ...

```
enum meses { JAN = 1, FEV, MAR, ABR, MAI, JUN, JUL, AGO, SET, OUT, NOV, DEZ };
```

Constantes do tipo "float" podem ser definidas por "#define's"

```
#define PI    3.14
#define SMALL 1e-9
#define TINY  1e-12
```

Constantes para caracteres e para *strings*

Um caractere é representado entre aspas simples: 'a', 'A', '3', '#', '\n'.

Uma sequência de caracteres, terminada por '\0' é chamada de *string* e é representada entre aspas duplas: "uma frase", "\toutra frase\n". A segunda frase é iniciada por um TAB ('\t') e terminada por um NEWLINE ('\n'): TABoutra fraseNEWLINE

Constantes para o tipo Boolean

As constantes TRUE e FALSE podem ser definidas com apelidos:

```
#define TRUE  (0==0)
#define FALSE (0!=0)
```

O pré-compilador substitui todas as ocorrências das palavras TRUE e FALSE pelas respectivas definições, e o compilador avalia as definições e substitui os resultados pelos valores adequados.

Outra alternativa é definir um "tipo enumerado":

```
enum boolean { FALSE TRUE };
```

O compilador atribui o valor inteiro 0 (zero) ao primeiro elemento da enumeração, e 1 (um) ao segundo, que são as definições da linguagem para falso (0) e verdadeiro (1).