−Tabela de conteúdos

Tipos Básicos

Para se criar variáveis em um programa C deve-se indicar para o compilador qual o tipo desta variável. Uma variável pode ter um tipo básico, intrínseco à linguagem C ou tipo estruturado, montado pelo programador. O programador pode *criar* novos tipos de dados, utilizando a declaração typedef.

A linguagem C define os seguintes tipos básicos de variáveis:

- **int** - Variável tipo inteira. Deve ser utilizado para se armazenar valor inteiro, com ou sem sinal.
- **char** - Variável do tipo caracteres. Servirá para se armazenar um único caractere.
- **float** - Para valores com casas decimais (reais) deve-se utilizar este tipo. Ele pode armazenar números reais com até 6 dígitos significativos.
- **double** - É o mesmo que o anterior, só que pode armazenar mais dígitos, dando uma precisão maior nos cálculos com casas decimais.

O tipo **void** deve ser utilizado não para variáveis, mas sim para indicar que uma função não tem nenhum valor retornado ou não possui nenhum parâmetro de entrada.

A padronização Ansi C 99 especifica ainda mais 2 tipos de variáveis:

- **_Bool** - Variável tipo booleana (verdadeiro ou falso). Ressalta-se que na linguagem C, em comparações lógicas, 0 (zero) é considerado falso e diferente de 0 (zero) é considerado verdadeiro
- **complex** - Variável para trabalhar com valores complexos (raízes imaginárias, por exemplo).

Abrangência e Modificadores de Tipo

A linguagem ANSI C determina para cada tipo intrínseco um certo tamanho em *bytes*. Este tamanho irá determinar a escala de valores que pode ser colocada dentro de um determinado tipo.

Abaixo estão os limites de valores aceitos por cada tipo e o seu tamanho ocupado na memória. Também nesta tabela está especificado o formato que deve ser utilizado para ler/imprimir os tipos de dados com a função scanf e printf

Tipo	Número de bits	Formato para leitura/impressão	Início	Fim
_Bool	8	Não tem (pode-se utilizar %d)	0	1
char	8	%c	-128	127
unsigned char	8	%c	0	255
signed char	8	%c	-128	127
int	32	%i	-2.147.483.648	2.147.483.647
unsigned int	32	%u	0	4.294.967.295
signed int	32	%i	-2.147.483.648	2.147.483.647
short int	16	%hi	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi	-32.768	32.767
long int	32	%li	-2.147.483.648	2.147.483.647
signed long int	32	%li	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
float	32	%f	3,4E-38	3.4E+38
double	64	%lf	1,7E-308	1,7E+308
long double	80	%Lf	3,4E-4932	3,4E+4932

Estes limites podem ser verificados no arquivo limits.h do pacote C e são válidos para plataformas de 32 bits. Em platarmos de 16 bits, o int era definido com 16 bits (o equivalente a **`short int`** na plataforma de 32 bits). Em plataformas de 64 bits:

Tipo	Número de bits	Formato para leitura/impressão	Início	Fim

Tipo	Número de bits	Formato para leitura/impressão	Início	Fim
long	64	%li	-9223372036854775806	9223372036854775807
unsigned long	64	%lu	0	18446744073709551615

O tamanho de uma varíavel ou de um tipo pode ser verificado com o operador sizeof.

Pode-se modificar o comportamento de um tipo básico, tanto no tamanho (espaço em memória) como no seu sinal (positivo ou negativo). Os modificadores de sinais indicam para o compilador considerar ou não valores negativos para o tipo inteiro. Apesar de existir a palavra **`signed`** ela é padrão não precisando ser colocada, mas pode sofrer influência do sistema operacional.

A palavra **`unsigned`** indica para o compilador não considerar o bit de sinal, estendendo assim o limite da variável inteira.

Pode-se modificar o tamanho das variáveis do tipo **`int`** para que ele ocupe somente dois bytes na memória, reduzindo assim o limite de abrangência para –32768 a +32767. Para isto coloca-se o modificador **`short`** na definição da variável, indicando que será utilizado somente dois bytes de tamanho. Devem-se tomar alguns cuidados com a portabilidade, pois num sistema a variável pode ser considerada **`signed`** e em outros **`unsigned`**, em alguns sistemas operacionais (variações de Unix, OS/2 da IBM, etc.) as variáveis são definas por padrão com **`unsigned`** (sem sinal), em outros como **`signed`** (com sinal).

Conversão de Tipos

De uma forma geral pode-se realizar a conversão de um tipo para outro da linguagem C utilizando o que se chama de ***typecast***. Esta técnica é muito utilizada para se melhorar o entendimento de alguns trechos de programas. Outras vezes utiliza-se o *typecast* para compatibilizar um determinado tipo de um parâmetro na chamada de uma função para o tipo do parâmetro esperado por aquela função.

A sintaxe do *typecasting* é: `(tipo) valor_constante` `(tipo) variável`

No primeiro formato o compilador irá realizar a transformação da constante indicada para o tipo indicado entre parênteses durante o processo de compilação. No segundo formato o compilador irá gerar o código adequado para que a conversão ocorra em tempo de execução.

Exemplos:

```
int iASCII = (int) 'E'; /* Código ASCII do 'E' */


/* converter o resultado de uma divisão para inteiro */
short int si;
float f;
int i;
i = (int) (f/si);
```

Formatadores de Tipos

Para cada variável colocada no comando printf ou correlatos deve-se indicar qual o formato desejado de saída. Isto é feito colocando-se o caractere '%' seguido de uma letra dentro do texto informado como primeiro parâmetro da função printf.

As funções não fazem nenhuma verificação entre o tipo real da variável e o caractere formatador indicado. Também não é feita à verificação do número correto de formatadores, um para cada variável. Quando isto acontecer, só será percebido quando da execução do programa, gerando resultados imprevisíveis.

Formato	Tipo da Variável	Conversão realizada
%c	caracteres	char, short int, int, long int
%d	inteiros	int, short int, long int
%e	Ponto flutuante, notação científica	float, double
%f	Ponto flutuante, notação decimal	float, double
%lf	Ponto flutuante; notação decimal	double
%g	O mais curto de %e ou %f	float, double
%o	Saída em octal	int, short int, long int, char

Formato	Tipo da Variável	Conversão realizada
%s	String	char *, char []
%u	Inteiro sem sinal	unsigned int, unsigned short int, unsigned long int
%x	Saída em hexadecimal (0 a f)	int, short int, long int, char
%X	Saída em hexadecimal (0 a F)	int, short int, long int, char
%ld	Saída em decimal longo	Usado quando long int e int possuem tamanhos diferentes

Indicando o Tamanho

Quando é feita a saída do valor de uma variável, além de se especificar o tipo (formato) que deve ser mostrado, pode-se indicar o tamanho da saída.

A indicação do tamanho depende do tipo da variável. Para os números inteiros (int, short int, long int, unsigned int, unsigned short int e unsigned long int) a especificação do tamanho tem a seguinte sintaxe:

```
% [tamanho].[qtd_dígitos]d
```

Onde:

- tamamanho - Indica o tamanho mínimo que deve ser colocado na saída caso o número possua quantidade menor de dígitos. Se o número possuir quantidade de dígitos maior que o valor, o número não será truncado.
- qtd_dígitos - Quantidade de dígitos que deve ser mostrada. Caso o número possua quantidade menor que o indicado, serão colocados zeros à esquerda até se completar o tamanho indicado.

Para os números reais (float e double), têm-se o seguinte formato:

```
% [tamanho].[casas_decimais]f
```

Onde:

- tamamanho - É o mesmo que descrito acima para os números inteiros. Vale completar que neste tamanho estão consideradas as casas decimais inclusive.
- casas_decimais - Número de casas decimais que devem ser mostradas. Caso o número possua número menor de decimais, o número será completado com zeros até o tamanho indicado. Se o número possuir um número de casas decimais maior que o indica a saída será truncada para o tamanho indicado.

Para as variáveis do tipo string pode-se indicar o tamanho mínimo e máximo a ser mostrado através da seguinte sintaxe:

```
%[tamanho].[tamanho_máximo]s
```

Neste caso se a string possuir tamanho menor que o indicado a saída será completada com brancos à esquerda.

Veja o exemplo da função printf.

Outros tipos

O sistema operacional pode definir alguns tipos e de dados também, estas definições podem ser encontradas no arquivos `sys/types.h` do seu compilador.

Alguns exemplos:

- `pid_t` para número de PID (veja a função fork);
- `pthread_t` para números de thread (veja as threads POSIX;
- `ssize_t` para definir tamanhos de blocos (vejas as funções para tratamento de rede)

Estas definições são utilizadas para garantir a portabilidade entre as plataformas 32 e 64 bits.

— Marcos Laureano [mailto:marcos@laureano.eti.br] 2009/02/14 12:36