



Universidade Estadual de Campinas – UNICAMP

Centro Superior de Educação Tecnológica – CESET

**Linguagem C - vol. 1**  
Prof.: Marco Antonio Garcia de Carvalho

Julho 2003  
Campinas, SP - Brasil

# Sumário

<b>1</b>	<b>Introdução</b>	<b>2</b>
1.1	Breve Histórico . . . . .	2
1.2	Características . . . . .	2
1.3	Estrutura básica de um programa . . . . .	3
1.4	Diretivas . . . . .	3
1.4.1	Diretiva <i>include</i> . . . . .	3
1.4.2	Diretiva <i>define</i> . . . . .	3
1.5	Palavras reservadas . . . . .	4
1.6	O primeiro programa (e o segundo) . . . . .	4
1.7	Tipos de dados em C . . . . .	5
1.7.1	Tipos de variáveis . . . . .	5
1.7.2	Declaração de variáveis . . . . .	6
1.7.3	Ponteiros . . . . .	6
<b>2</b>	<b>Operadores</b>	<b>8</b>
2.1	Atribuição . . . . .	8
2.2	Aritméticos . . . . .	8
2.3	Incremento e decremento . . . . .	8
2.4	Arimético de atribuição . . . . .	9
2.5	Relacionais . . . . .	9
2.6	Lógicos . . . . .	9
2.7	Condiciona ternário . . . . .	9
2.8	Tabela de precedência . . . . .	10
<b>3</b>	<b>Entrada - Saída</b>	<b>11</b>
3.1	A função <i>printf()</i> . . . . .	11
3.2	A função <i>scanf()</i> . . . . .	11
3.3	Outras funções de entrada de dados . . . . .	11
<b>4</b>	<b>Laços</b>	<b>12</b>
4.1	O laço <i>for</i> . . . . .	12
4.2	O laço <i>while</i> . . . . .	13
4.3	O laço <i>do while</i> . . . . .	13
<b>5</b>	<b>Estruturas de seleção</b>	<b>14</b>
5.1	O comando <i>if</i> . . . . .	14
5.2	O comando <i>switch</i> . . . . .	15
	<b>Bibliografia</b>	<b>17</b>

# 1 Introdução

## 1.1 Breve Histórico

C é uma linguagem de programação cujas instruções consistem de termos semelhantes a expressões algébricas, acrescidas de certas palavras chaves em inglês.

C foi desenvolvida a partir de duas linguagens anteriores, a BCPL e B, criadas no final da década de 60. Ambas as linguagens eram definidas sem tipo, ou seja, qualquer dado ocupava um mesmo espaço na memória.

A linguagem C foi criada por Dennis Ritchie e Brian W. Kernighan, em 1972, no centro de pesquisa da Bell Laboratories. Sua primeira utilização importante foi a reescrita do sistema Unix. Em meados da década de 70, o Unix foi liberado para utilização nas universidades, dando o impulso para o sucesso da linguagem.

ANSI C (*American National Standards Institute*) é a versão da linguagem C padronizada em 1989 nos EUA como também no mundo através da International Standards Organization — ISO.

C++ é uma linguagem derivada da linguagem C. O conjunto de instruções que fazem parte da linguagem C também é parte de C++. Os elementos principais que foram adicionados à linguagem C para dar origem a C++ consiste na idéia de programação orientada ao objeto. Qualquer programa em C compilado no padrão ANSI também pode ser compilado em C++, ou seja, C e C++ são compatíveis.

## 1.2 Características

Os sistemas em C normalmente possuem 3 partes: o ambiente, a linguagem e a biblioteca padrão. As funções da biblioteca não fazem parte da linguagem C, mas realizam operações importantes tais como entrada e saída de dados. As principais características da linguagem C são:

- Portabilidade
- Modularidade
- Compilação separada
- Recursos de baixo nível
- Confiabilidade (geração de código eficiente)
- Simplicidade

A geração de um programa executável a partir do programa fonte obedece a uma seqüência de operações, resumida a seguir:

1. EDITOR – módulo fonte em C. O programador digita um programa em um editor. Os pacotes de softwares C normalmente vêm com editor embutido.
2. PRÉ-PROCESSADOR – novo fonte expandido. Responsável por incluir novos arquivos e substituir símbolos especiais por texto.
3. COMPILADOR – módulo em linguagem de máquina ou *código objeto*.
4. LIGADOR (LINKER) – criação do programa executável. Faz a ligação do *código objeto* com o código de funções para produzir um código executável.

### 1.3 Estrutura básica de um programa

A unidade fundamental de programa C são as funções, onde um programa consiste de uma ou mais funções.

*definições de pré-processamento*

*declaração das variáveis globais*

```
{  
instrução_1;  
instrução_2;  
:  
instrução_n;  
}
```

Toda função C deve começar com uma chave de abertura de bloco e deve terminar com uma chave de fechamento de bloco. Todo programa em C deve ter uma função chamada *main()*, que é a primeira função a ser executada. O programa termina sua execução quando é encontrada a chave que fecha o corpo da função *main()*.

As definições de pré-processamento (diretivas ou arquivos de inclusão) provocam a inclusão de um outro arquivo em nosso programa fonte.

### 1.4 Diretivas

#### 1.4.1 Diretiva *include*

Copia o conteúdo do arquivo especificado para dentro do programa. A sintaxe é a seguinte:

```
# include <nome do arquivo>
```

É usada para arquivos do tipo *.h* (*header*) que contém funções da biblioteca padrão, definição de tipos e macros. O arquivo é procurado em um diretório padrão de headers do compilador. Quando se utiliza aspas duplas, ao invés de <>, o arquivo é procurado no diretório atual.

Alguns dos principais arquivos de cabeçalhos do C são dados abaixo.

Arquivo	Descrição
stdio.h	funções de entrada e saída
stdlib.h	funções de uso genérico
string.h	funções de tratamento de strings
math.h	funções matemáticas

Tabela 1: Alguns arquivos de cabeçalhos.

#### 1.4.2 Diretiva *define*

A diretiva *define* substitui toda ocorrência de um nome especificado por um valor determinado. É definida desta forma:

```
#define nome valor
```

Também é conhecida como substituição de macro e por convenção, todo nome deve ser escrito em letras maiúsculas.

- Ex.:  
*#define* PI 3.14159

## 1.5 Palavras reservadas

Palavras chaves ou palavras reservadas possuem um significado especial para o compilador C, de modo que o programador não deve utilizá-las como identificadores ou nomes de variáveis (lembrando que C diferencia maiúsculas de minúsculas).

O conjunto completo de palavras reservadas do ANSI C é mostrado a seguir.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Tabela 2: Palavras reservadas do C.

## 1.6 O primeiro programa (e o segundo)

Um dos primeiros programas presentes na maioria dos livros de C é o programa que escreve uma simples mensagem na tela, como exemplificado a seguir.

```
#include <stdio.h>

main()
{
    printf ("Primeiro Programa\n");
}
```

Notas importantes:

- Deve existir em todo programa uma função *main()*.
- Toda função deve começar e terminar com uma chave.
- Você pode inserir espaços em brancos e linhas à vontade em seus programas, com exceção para a diretiva *include*.
- Toda instrução C termina com um ponto-e-vírgula.
- Os arquivos fonte em C são gravados com a extensão **.c**.
- Sempre que possível deve-se incluir comentários para documentar os programas e melhorar sua legibilidade.

Um programa igualmente simples e que contém comandos de entrada e saídas de informações é dado abaixo.

```
/* Programa da soma de dois números */
#include <stdio.h>

main()
{
    int inteiro1, inteiro2, soma;
    printf ("Entre com o primeiro inteiro\n"); /* Lê um número */
    scanf ("%d", &inteiro1);
```

```
printf ("Entre com o segundo inteiro\n"); /* Lê um número */
scanf ("%d", &inteiro2);
soma = inteiro1 + inteiro2; /* calcula a soma */
printf ("A soma eh %d\n", soma); /* escreve o resultado */
}
```

A barra invertida (*backslash*) é chamada de caractere de escape. Ela indica que o **printf** deve fazer algo diferente do normal, quando combinado com o caractere imediatamente posterior. A tabela abaixo apresenta as principais seqüências de escape.

Seqüência de escape	Descrição
\n	Nova linha
\t	Tabulação horizontal
\r	<i>Carriage return</i>
\a	Soa um bip (sinal sonoro)

Tabela 3: Seqüências comuns de escape.

## 1.7 Tipos de dados em C

Variáveis e constantes são o aspecto fundamental de qualquer linguagem de programação; significam um espaço em memória reservado para armazenar um certo tipo de dado, tendo um nome para referenciar o seu conteúdo. Uma variável é um espaço em memória que pode conter, a cada tempo, valores diferentes.

### 1.7.1 Tipos de variáveis

Tipo de variável diz respeito ao tamanho de memória e à forma de armazenamento. Existem cinco tipos básicos, como apresenta a Tabela 4 abaixo.

Tipo	Bit	Bytes	Faixa de Abrangência
char	8	1	-128 até 127
int	16	2	-32768 até 32767
float	32	4	3.4e-38 até 3.4e+38
double	64	8	1.7e-308 até 1.7e+308
void	0	0	nenhum valor

Tabela 4: Tipos de variáveis em C.

A variável tipo **float** (ou ponto flutuante) corresponde ao número real. Para armazenar essas variáveis, o computador separa o número em duas partes: a mantissa e o expoente. Os tipos de dados básicos podem ser acompanhados por modificadores, a seguir:

- long
- short
- unsigned

O modificador pode ser utilizado sem que se especifique o tipo. Neste caso, é assumido por *default* o tipo **int**. Por exemplo, o tipo **unsigned char**, 8 bits, teria uma faixa de abrangência de 0 a 255.

Quando se utiliza o tipo de dado *caracter*, aspas simples servem para representar um único caracter em ASCII e aspas duplas, uma cadeia de caracteres. Em geral, números negativos são representados usando-se a notação de *complemento de dois*.

### 1.7.2 Declaração de variáveis

C não funciona se você não declarar suas variáveis. As variáveis são declaradas em qualquer lugar do programa, desde que antes de serem utilizadas. Uma declaração de variável consiste no nome de um tipo seguido do nome da variável (identificador), seguido de ponto-e-vírgula. Por exemplo:

```
int contador;  
float acumulador;
```

Inicializar uma variável significa atribuir um valor a ela na mesma instrução de sua declaração. Utiliza-se o operador de atribuição (=) para a definição do valor inicial. Exemplo:

```
int contador = 0;  
float tempo = 30.00;
```

Você pode usar quantos caracteres quiser para um nome de variável, sendo o primeiro caractere obrigatoriamente uma letra ou o sublinhado. Somente os 32 primeiros caracteres de um nome de variável são significativos e não é permitido o uso de uma palavra-chave da linguagem. Em C, letras maiúsculas e minúsculas são diferentes.

Exemplo de um programa com definição de variáveis:

```
#include <stdio.h>  
main()  
{  
  unsigned j = 65000;  
  printf ("Variável unsigned = %d\n",j);  
  int i = j;  
  printf ("Variável int = %d,\n",i);  
}
```

Qual será a saída deste programa?

### 1.7.3 Ponteiros

A linguagem permite referenciar a posição de um identificador bem como o seu conteúdo. Se  $x$  é declarado como:

```
int x;
```

Então  $\&x$  refere-se à posição de memória (endereço) reservada que conterà o conteúdo de  $x$  (uma seqüência de bits que representará um valor inteiro). É possível declarar uma variável cujo conteúdo seja posições de memória (um ponteiro):

```
int *pi;  
float *pf;
```

Um exemplo de manipulação:  $pi = \&x$ ;

Um uso freqüente de ponteiros é na passagem de parâmetros em uma função (passagem por referência).

Ao longo de um programa, pode-se referenciar o conteúdo do endereço indicado pelo ponteiro, através da notação  $*pi$ . Por exemplo:

```
int x = 1, y;  
int *pi;  
  
...  
pi = &y;  
Y = x;  
  
Qual o valor de *pi? R:1
```



## 2 Operadores

### 2.1 Atribuição

É o sinal de igual (=). Atribui a expressão da direita à variável à sua esquerda. Em C pode ocorrer atribuições múltiplas.

**Ex.:**

```
y = 3;
y = x = 3;
y = ( x = 3 );
```

### 2.2 Aritméticos

Existem operadores denominados binários (trabalham com dois operandos) e unários (funcionam com um operando).

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo (resto da divisão do inteiro à sua esquerda pelo inteiro à sua direita)

Tabela 5: Operadores aritméticos binários.

-	Menos unário (mult. o valor ou variável por -1)
---	---

Tabela 6: Operadores aritméticos unários.

### 2.3 Incremento e decremento

Incrementam (++) ou decrementam (-) a variável operando de 1. Podem ser prefixado ou pós-fixado.

**Ex.:**

```
x = x + 1;
++ x; ou x++; adiciona 1 a x
```

## 2.4 Arimético de atribuição

Combinam operações aritméticas com atribuições. Compactam as operações. Segue a regra:

**variável operação= expressão**

**Ex.:**

```
i += 2; equivale a i = i + 2
x *= y + 1 equivale a x = x * (y + 1)
t /= 2.5; equivale a t = t/2.5
p %= 5; equivale a p = p % 5
```

## 2.5 Relacionais

São eles:

> Maior, >= Maior ou igual, < Menor, <= Menor ou igual, == igual, != Diferente

Os operadores relacionais têm precedência menor que os aritméticos.

## 2.6 Lógicos

São três operadores:

Operador	Descrição
&& lógico E ( <i>and</i> )	Resulta 1 ( <i>V</i> ) somente se as duas expressões forem verdadeiras
lógico ou ( <i>or</i> )	Resulta 0 ( <i>F</i> ) somente se as duas expressões forem falsas
! lógico não ( <i>not</i> )	Resulta 1 ( <i>V</i> ) somente se a expressão for falsa

Tabela 7: Operadores lógicos *and*, *or* e *not*.

## 2.7 Condicional ternário

É o único que opera sobre três expressões.

**exp1 ? exp2 : exp3**

*exp1* é avaliada primeiro. Se for verdadeira, *exp2* é avaliada e seu resultado é o valor da expressão como um todo. Se *exp1* for zero (falso), *exp3* é avaliada e será o valor da expressão condicional.

• **Ex.:**

*max* = (*a* > *b*)? *a* : *b*; A variável que contém o maior valor numérico entre *a* e *b* será atribuída a *max*.

## 2.8 Tabela de precedência

O nível de precedência dos operadores é avaliado da esquerda para a direita. Os parênteses podem ser utilizados para alterar essa ordem, sendo indicado seu uso a fim de tornar o programa (a expressão) mais legível.

A tabela abaixo mostra a precedência dos operadores vistos até agora.

Maior	()
	!, ++, --, -(unário)
	*, /, %
	+, -
	==, !=
	&&
	?:
Menor	+=, -=, *=, /=

Tabela 8: Ordem de precedência dos operadores em C.

## 3 Entrada - Saída

### 3.1 A função *printf()*

Faz parte da biblioteca padrão da linguagem C e é responsável em escrever e formatar os dados de saída. Toda chamada a *printf()* contém uma string de controle de formato que descreve o formato de saída. A função *printf()* está definida em *stdio.h* e tem a seguinte forma:

```
printf(string de controle de formato , outros argumentos);
```

A string de controle de formato (que fica entre aspas duplas) descreve o formato da saída; outros argumentos (opcionais) correspondem a cada especificação de conversão na string de controle de formato.

Os principais formatos são dados na tabela abaixo.

Especificador	Descrição
% d	Exibe um inteiro decimal
% i	Exibe um inteiro decimal com sinal
% f	Exibe valores de ponto flutuante
% e	Exibe valores de ponto flutuante em notação exponencial
% c	Exibe caracteres
% s	Exibe strings

Tabela 9: Principais formatos.

### 3.2 A função *scanf()*

Apropriada para entrada de valores numéricas (*int*, *float*, *double*). A formatação dos dados é semelhante à função *printf*, porém deve-se utilizar obrigatoriamente o símbolo & antes do nome da variável a ser lida. Está definida em *stdio.h*. Veja um exemplo de sintaxe:

```
int n;
```

```
printf ("Informe o número de termos: \n"); scanf ("%d", &n);
```

### 3.3 Outras funções de entrada de dados

- **getchar()** — Lê um caracter e retorna-o. Deve-se teclar ENTER após o caracter digitado.

Por exemplo:

```
char ch;
```

```
ch = getchar();
```

```
printf ("O caracter digitado foi %c \n", ch);
```

Pode ser usada sem a necessidade de retornar valor, simplesmente para provocar uma pausa no programa.

- **gets()** — Lê strings.

```
gets(string_digitada);
```

Obs.: As funções acima também estão definidas em *stdio.h*. Devido forma como a função *scanf* trata o buffer de entrada, quando ela for usada com as demais funções deve-se limpar o buffer com a função **fflush(stdin)**.

## 4 Laços

Existem três estruturas de laços: *for*, *while*, *do-while*.

### 4.1 O laço *for*

É utilizado quando conhecemos o número de iterações a serem utilizadas. Consiste na estrutura abaixo:

```
for ( inicialização ; teste ; incremento )
instrução;
```

- inicialização : executada uma única vez antes do laço ser iniciado.
- teste: avalia como  $V$  ou  $F$ .
- incremento: executada imediatamente após a execução do corpo do laço.
- Possibilidade de declaração de variáveis no próprio *for*.
- O operador vírgula (mais de uma instrução nas expressões do *for*).
- Usando chamada a funções (*getch()*).
- Omitindo expressões (; permanece). Se omitido teste, considerar sempre  $V$ .
- Caso tenha-se mais de uma instrução, delimitar o bloco através de chaves. Uma variável declarada em um bloco não é visível fora dele.
- Laços *for* aninhados (um *for* dentro de outro).
- **Exercícios**

1. Calcular o valor do somatório  $S$  abaixo para um número de termos especificado pelo usuário.

$$S = 2 + 6 + 12 + 20 + \dots + (n + 1)n \quad (1)$$

```
#include <stdio.h>
main()
{
int n; float S=0.0;
printf ("Informe o número de termos: \n"); scanf("%d", &n);
for(int i=1; i<=n; i++)
S = S + i * (i + 1);
printf ("O valor de S para %d termos eh %f \n", n , S);
fflush(stdin); getchar();
}
```

2. Calcular uma tabela com o quadrado dos números naturais de 1 a 100.

## 4.2 O laço *while*

Apropriado em situações onde o laço pode ser terminado inesperadamente, por condições desenvolvidas dentro do laço. Sintaxe:

```
Expressão de inicialização;  
while (teste)  
{  
  instruções;  
  expressão de incremento;  
}
```

O programa a seguir apresenta um exemplo simples de utilização do *while* na contagem de valores inteiros.

```
/* Programa contagem */  
#include <stdio.h>  
main()  
{  
  int i=0;  
  while(i < 10)  
  {  
    printf ("%d", i);  
    i --;  
  }  
}
```

## 4.3 O laço *do while*

É utilizado em situações em que é necessário executar o corpo do laço uma primeira vez e depois avaliar a expressão de teste e criar um ciclo repetido. Sintaxe:

```
do  
{  
  instrução;  
  instrução;  
} while (teste);
```

## 5 Estruturas de seleção

### 5.1 O comando *if*

Sintaxe:

```
if ( condição )
instrução 1;
else
instrução 2;
```

Se condição for verdadeira, é executada a instrução 1; caso contrário, é executada a instrução 2. Múltiplas instruções devem ser delimitadas por chaves em um bloco. O uso do *else* é opcional. O programa a seguir determina se um número é par ou é ímpar.

```
#include <stdio.h>
int main(void)
{
int num;
printf ("Digite um numero:\n");
scanf("%d", & num);
if(num%2==0){
printf("%d eh par\n", num);
printf("A metade eh %d \n", num/2);
}
else
printf("%d eh impar\n", num);
fflush(stdin); getchar();
return(0);
}
```

#### • Exercícios

1. Ler três valores  $A$ ,  $B$  e  $C$ , referentes aos lados de um triângulo. Verificar se estes valores formam um triângulo: caso positivo, determinar se o mesmo é equilátero, isósceles ou escaleno; caso contrário, escreva uma mensagem de erro.
2. Qual a menor quantidade de *ifs* que deve ser usada para determinar o maior entre três números de valores diferentes  $x$ ,  $y$  e  $z$ ?
3. Calcular o valor do somatório  $S$  abaixo para um número de termos  $n$  especificado pelo usuário.

$$S = 1 - 2 + 3 - 4 + 5 - 6 + \dots \quad (2)$$

```

/* Programa triângulo */
#include <stdio.h>
main()
{
float A, B, C;
printf ("Informe o valor do lado A:\n");scanf("%f", &A);
printf ("Informe o valor do lado B:\n");scanf("%f", &B);
printf ("Informe o valor do lado C:\n");scanf("%f", &C);
if(A < B + C && B < A + C && C < A + B)
if(A == B && B == C)
printf("ABC formam um triângulo equilátero\n");
else
if(A == B || A == C || B == C)
printf("ABC formam um triângulo isósceles\n");
else printf("ABC formam um triângulo escaleno\n");
else
printf("ABC não formam um triângulo\n");
fflush(stdin); getchar();
}

```

## 5.2 O comando *switch*

Sintaxe:

```

switch ( expressão ) {
case constante1:
instruções;
break;
case constante2:
instruções;
break;
case constanteN:
instruções;
break;
default:
instrução;
}

```

Primeiro é avaliada a expressão após o comando *switch*. São executados as instruções seguintes ao *case* onde coincidir o valor da constante. Após encontrar o comando *break*, o restante do *switch* é desprezado. Caso nenhum valor de *case* coincida com a expressão, então é executada as instruções após *default* (seu uso é opcional).

O exemplo a seguir apresenta um programa para calcular o valor da conta em uma lanchonete que vende pizza, hambúrguer e cachorro quente.

```

#include <stdio.h>
int main(void)
{
float total=0.0;
int escolha; printf ("1 - Pizza\n");
printf ("2 - Hamburger\n");
printf ("3 - Cachorro quente\n");
printf ("0 - FIM\n");
do {
printf ("Faca sua escolha:\n");
scanf("%d", & escolha);
switch ( escolha ) {
case 1:
printf ("Voce escolheu pizza.\n");
total +=15;
break;
case 2:
printf ("Voce escolheu hamburger.\n");
total +=5.5;
break;
case 3:
printf ("Voce escolheu cachorro quente.\n");
total +=2.5;
break;
default:
printf ("OPCAO INVALIDA.\n");
}
} while(escolha!=0);
printf ("Total a pagar = R$ %10.2f\n", total);
fflush(stdin); getchar();
return(0);
}

```

## Referências

- [1] Schildt, H. C Completo e Total. São Paulo: Makron Books, 1996.
- [2] Deitel, H. M., Deitel, P. J. Como Programar em C. Rio de Janeiro: LTC, 1999.
- [3] Arnush, C. Teach Yourself Turbo C++ 4.5 For Windows in 21 Days. Indianapolis: Sams Publishing, 1998.
- [4] Manzano, J. A. N. G. Linguagem C - Estudo Dirigido. São Paulo: Érica, 1997.
- [5] CÔRTEZ, P. L. Turbo C: Ferramentas & Utilitários, vol. 1. São Paulo: Érica, 1999.
- [6] Lehmann, A.H., Trevisan, R. Turbo C. Guia de Referência Rápida de todas as Funções. São Paulo: Érica, 1990.