LINGUAGEM C

A **linguagem C** é uma linguagem de alto nível, genérica. Foi desenvolvida por programadores para programadores tendo como meta características de *flexibilidade* e *portabilidade*.

O C é uma linguagem que nasceu juntamente com o advento da teoria de linguagem estruturada e do computador pessoal. Assim tornou-se rapidamente uma linguagem "popular" entre os programadores. O C foi usado para desenvolver o sistema operacional UNIX, e hoje esta sendo usada para desenvolver novas linguagens, entre elas a linguagem C++ e Java.

Entre as principais características da linguagem C, citamos:

- O C é uma linguagem de alto nível com uma sintaxe bastante estruturada e flexível tornando sua programação bastante simplificada.

- Programas em C são compilados, gerando programas executáveis.

- O C compartilha recursos tanto de alto quanto de baixo nível, pois permite acesso e programação direta do microprocessador. Com isto, rotinas cuja dependência do tempo é crítica, podem ser facilmente implementadas usando instruções em Assembly. Por esta razão o C é a linguagem preferida dos programadores de aplicativos.

- O C é uma linguagem estruturalmente simples e de grande portabilidade. O compilador C gera códigos mais enxutos e velozes do que muitas outras linguagens.

- Embora estruturalmente simples (poucas funções intrínsecas) o C não perde funcionalidade pois permite a inclusão de uma farta quantidade de rotinas do usuário. Os fabricantes de compiladores fornecem uma ampla variedade de rotinas pré-compiladas em bibliotecas.

HISTÓRICO

A primeira versão de **C** foi criada por *Dennis Ritchie* em **1972** nos laboratórios *Bell Telephones, Inc* para ser incluído como um dos softwares a ser distribuído juntamente com o sistema operacional Unix do computador PDP-11, na equipe certificada por *Ken Thompson*.

Ao ponto de vista técnico, o surgimento do **C** iniciou com a linguagem **ALGOL 60**, definida em **1960**. **ALGOL** era uma linguagem de alto nível, que permitia ao programador trabalhar "*longe da máquina*", sem se preocupar com os aspectos de como cada comando ou dado era armazenado ou processado. Foi criado para substituir o **FORTRAN**. **ALGOL** não teve sucesso, talvez por tentar ser de muito alto

nível em uma época em que a maioria dos sistemas operacionais exigiam do usuário um grande conhecimento de hardware.

Em **1967** surgiu **CPL** (*Combined Programming Language*) nas universidades de Londres e Cambridge com o objetivo, segundo a equipe do projeto, de "*trazer ALGOL à terra*", ou "*manter contato com a realidade de um computador real*". Da mesma forma de **ALGOL**, **CPL** não foi bem aceita, em especial pelos projetistas de sistemas operacionais que a consideravam difícil de implementar.

Ainda em **1967**, em Cambridge, *Martin Richards* criou o **BCPL** (*Basic CPL*), uma simplificação do **CPL**, tentando manter apenas as "*boas coisas do CPL*".

Em **1970**, *Ken Thompson*, chefe da equipe que projetou o UNIX para o PDP11 do *Bell Labs*, implementou um compilador para uma versão mais reduzida do **CPL**. Batizou a linguagem de **B**.

Tanto **BCPL** quanto **B** mostravam-se muito limitadas, prestando-se apenas para certas classes de problemas. Isto se fez sentir especialmente na primeira versão do PDP11, lançado no mercado em 1971. Um dos fatores que levou à isto foi a intenção do grupo responsável pelo UNIX de reescrevê-lo todo em uma linguagem de alto nível, e para isto **B** era considerado lenta.

Estes problemas levaram a que o projetista *Dennis Ritchie*, do *Bell Labs*, fosse encarregado de projetar uma nova linguagem, sucessora do **B**, que viria então, a ser chamada de **C**.

A linguagem **C** buscou manter o "*contato com o computador real*" e ainda sim dar ao programador novas condições para o desenvolvimento de programas em áreas diversas, como comercial, científica e de engenharia.

Por aproximadamente 10 anos a sintaxe (formato) tida como padrão da linguagem **C** foi aquela fornecida com o UNIX versão 5.0 do *Bell Labs*. A principal documentação deste padrão encontra-se na publicação "*The C Programming Language*", de *Brian Kernighan* e *Dennis Ritchie* (K&R), tida como a "*bíblia da linguagem C*".

O mais interessante desta versão de **C** era que os programas-fonte criados para rodar em um tipo de computador podiam ser transportados e recompilados em outros sem grandes problemas. A esta característica dá-se o nome de *portabilidade*. Com ela, uma empresa que desenvolve um programa pode fazê-lo rodar em diferentes computadores sem ter um elevado custo a cada vez que isto for feito.

Em **1985**, **ANSI** (*American National Standards Institute*) estabeleceu um padrão oficial de **C** o chamado "**C ANSI**".

Em **1990**, a *Borland International Co*, fabricante de compiladores profissionais escolhe o **C** e o **Pascal** como linguagens de trabalho para o seu *Integrated Development Enviroment* (Ambiente Integrado de Desenvolvimento) e surge o **Turbo C**.

Em **1992**, o **C** se torna ponto de concordância entre teóricos do desenvolvimento da teoria de *Object Oriented Programming* (programação orientada a objetos) e surge o **C++**.

ESTRUTURA DE UM PROGRAMA EM C

Um programa em C é constituído de:

- Diretivas de compilação;
- Definições de tipos;
- Protótipos de funções;
- Funções;
- Comentários.

Programa exemplo:

```
#include <stdio.h>
int a;
main () {
    a = 10;
    printf("Valor de a: %d.", a);
}
```

em que

`#include <stdio.h>` é uma diretiva de compilação;
`int a` indica a criação de uma variável numérica inteira (`int`) de nome `a`;
`main()` é a função principal do programa.

DIRETIVAS DE COMPILAÇÃO

Em C, existem comandos que são processados durante a compilação do programa. Estes comandos são genericamente chamados de *diretivas de compilação*. Estes comandos informam ao compilador do C basicamente quais são as constantes simbólicas usadas no programa e quais bibliotecas devem ser anexadas ao programa executável. A diretiva `#include` diz ao compilador para incluir na compilação do programa outros arquivos. Geralmente estes arquivos contem bibliotecas de funções ou rotinas do usuário. A diretiva `#define` diz ao compilador quais são as constantes simbólicas usadas no programa.

SINTAXE

A sintaxe de um programa em linguagem C é a maneira de escrever o programa de forma que o compilador entenda. Esse conjunto de regras é específico de cada linguagem. Quando há a necessidade de utilizar outra linguagem de programação, a sintaxe dos comandos é modificada.

COMENTÁRIOS

Em C, comentários podem ser escritos em qualquer lugar do texto para facilitar a interpretação do algoritmo. Para que o comentário seja identificado como tal, ele deve ter um `/*` antes e um `*/` depois.

Desta forma, tudo que existir entre o intervalo `/* ... */` não será interpretado pelo compilador C, então pode ser escrito qualquer caractere ou frase a fim de auxiliar o programador ou quem está lendo o programa fonte.

```
/* comentário de um programa em C para um compilador C ANSI */

// comentário de um programa em C para o compilador C++
```

IDENTIFICADORES

Os identificadores são nomes simbólicos escolhidos pelo programador para representar endereços de memória, onde serão alocadas as informações, e objetos referenciados no programa.

Existem algumas regras para a utilização de identificadores:

- Todos os identificadores devem começar por `a...z` ou `A...Z` ou sublinhado (_).
- Os demais caracteres do identificador podem ser compostos de letras, números ou o próprio sublinhado. Nenhum outro caractere é permitido.
- Os primeiros 32 caracteres de um identificador são significativos.

Identificadores válidos	Identificadores inválidos

Obs: a linguagem C é caractere sensível (*case sensitive*) e interpreta letras minúsculas diferente de letras maiúsculas.

PALAVRAS RESERVADAS

Existem certos nomes que não podem ser usados como identificadores. São chamadas as *palavras reservadas* e são de uso restrito da linguagem C (comandos, estruturas, declarações, etc.).

O conjunto de palavras reservadas usadas em C é o seguinte:

```
asm         auto        break       case        cdecl       char
class       const       continue    _cs         default     delete
do          double      _ds         else        enum        _es
extern      _export     far         _fastcall   float       for
friend      goto        huge        if          inline      int
interrupt   _loadds     long        near        new         operator
pascal      private     protected   public      register    return
_saveregs   _seg        short       signed      sizeof      _ss
static      struct      switch      template    this        typedef
union       unsigned    virtual     void        volatile    while
```

CONSTANTES

A linguagem C possui quatro tipos básicos de constantes: ***inteiras***, ***de ponto flutuante***, ***caracteres*** e ***strings***. Constantes inteiras e de ponto flutuante representam números de um modo geral. Caracteres e *strings* representam letras e agrupamentos de letras (palavras).

- **Constantes inteiras**

Uma constante inteira é um número de valor inteiro. De uma forma geral, constantes inteiras são sequências de dígitos que representam números inteiros.

Números inteiros podem ser escritos no formato *decimal* (base 10), *hexadecimal* (base 16) ou *octal* (base 8).

Uma constante inteira *decimal* é formada por uma sequência de dígitos decimais: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`. Se a constante tiver dois ou mais dígitos, o primeiro não pode ser `0`. Na verdade, pode ser `0` mas o compilador considerará esta constante como *octal* e *não decimal*.

Exemplo:

Uma constante inteira *hexadecimal* é formada por uma sequência de dígitos decimais: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F` (ou `a, b, c, d, e, f`).

Uma constante hexadecimal deve começar por `0x`. Neste caso, os dígitos hexadecimais podem ser minúsculos ou maiúsculos.

Exemplo:

Uma constante inteira *octal* é formada por uma sequência de dígitos octais: `0, 1, 2, 3, 4, 5, 6, 7`. A constante octal deve ter o primeiro digito `0` para que o compilador a identifique como tal.

Exemplo:

- **Constantes de ponto flutuante**

Números reais (não inteiros) são representados em base 10, por números com um ponto decimal e (opcionalmente) um expoente.

Um número ponto flutuante *deve* ter um ponto decimal que não pode ser substituído por uma vírgula. Um número de ponto flutuante pode ser escrito em notação científica.

A forma de representação de um número real em C é bastante flexível.

Exemplo:

- **Constantes caracteres**

Uma constante caractere é uma letra ou símbolo colocado entre *aspas simples*.

Exemplo:

Embora sejam visualizados como letras e símbolos as constantes caracteres são armazenadas internamente pelo computador como um número inteiro entre 0 e 255. O caractere 'A' por exemplo, tem valor 65. Os valores numéricos dos caracteres estão padronizados em uma tabela chamada de ***A****merican* ***S****tandard* ***C****ode for* ***I****nformation* ***I****nterchange Table* ou simplesmente tabela *ASCII*.

Certos códigos de controle da tabela ASCII (como o *line feed*) ou caracteres especiais (como ') possuem representação especial no C. Esta representação chama-se **sequência de escape** representada por uma *barra invertida* (\) e um *caractere*. Sequencias de escape são interpretadas como caracteres simples. Abaixo segue uma lista das principais *sequencias de escape* usadas no C.

Controle/Caractere	**Sequencia de escape**	**Valor ASCII**
nulo (*null*)	\0	00
campainha (*bell*)	\a	07
retrocesso (*backspace*)	\b	08
tabulação horizontal	\t	09
nova linha (*new line*)	\n	10
tabulação vertical	\v	11
alimentação de folha (*form feed*)	\f	12
retorno de carro (*carriage return*)	\r	13
aspas (")	\"	34
apóstrofo (')	\'	39
interrogação (?)	\?	63
barra invertida (\)	\\	92

- **Constantes strings**

Uma constante *string* consiste de um conjunto de caracteres colocados entre *aspas duplas*.

Embora as instruções do C usem apenas os caracteres do conjunto padrão ASCII, as constantes caractere e *string* podem conter caracteres do conjunto estendido ASCII: é, ã, ç, ü, ...

Exemplo:

TIPOS DE DADOS

Em C, como na maioria das linguagens, os dados são divididos tipos: inteiro, real, caractere, etc. Esta divisão se deve basicamente ao número de *bytes* reservados para cada dado. Cada tipo de dado possui um intervalo de valores permitidos.

Segue uma lista dos tipos básicos de dados permitidos em C. Os tipos `char` e `int` são inteiros e os tipos `float` e `double` são de ponto flutuante.

Tipo	Tamanho	Intervalo	Uso
`char`	1 byte	-128 a 127	número pequeno e caractere ASCII
`int`	4 bytes	2147483648 e 2147483647	Inteiro, contador, controle de *loop*
`float`	4 bytes	3.4e-38 a 3.4e38	real (precisão de 7 dígitos)
`double`	8 bytes	1.7e-308 a 1.7e308	científico (precisão de 15 dígitos)

Tipos modificados

Além dos tipos de dados citados acima existem outros tipos de dados ditos modificados.

Em C existem modificadores de tipo em que é possível modificar o tamanho de armazenamento: o modificador `long`, o modificador `short`, o modificador `signed` e o modificador `unsigned`.

Tipicamente o modificador `long` aumenta o número de bytes usados para o registro do número. Por consequência o intervalo de validade do número fica aumentado significativamente. Pode ser usado com `int` e `long`.

O modificador short diminui o número de bytes usados para o registro do número. Só pode ser usado com `int`.

O modificador `unsigned`, usado somente em *inteiros*, permite que um bit usado para guardar o sinal do número seja usado para guardar o valor do número. Em consequência disto o intervalo do número fica dobrado, porém somente *permite o uso de números positivos*.

Tipo	Tamanho	Intervalo
`unsigned char`	1 byte	0 a 255
`unsigned int`	4 bytes	0 a 4294967295
`short int`	2 bytes	-32768 a 32767
`long int`	4 bytes	-2147483648 a 2147483647
`long double`	10 bytes	3.4e-4932 a 3.4e4932

Strings

Uma *string* é um conjunto ordenado de caracteres que pode ser armazenado sob forma de um vetor um ponteiro. Esta estrutura de dados será vista posteriormente. Por enquanto, basta saber como declarar e armazenar um conjunto caractere em uma variável.

Para declarar uma variável para receber um conjunto caractere devemos escrever:

Sintaxe: `char* var;`

Exemplo:

VARIÁVEIS

Nos computadores, dispositivos de memória permitem que as instruções a serem executadas e os dados a serem manipulados sejam armazenados temporariamente. Esta memória pode ser vista como um “armário” composto por muitas “gavetas”, que são chamadas variáveis.

O nome *variável* se deve ao fato de podermos substituir seu conteúdo quando necessário. Todas as variáveis devem receber um nome para sua identificação. Estes nomes são chamados *identificadores*.

Variáveis são objetos que podem ter diferentes valores durante a execução do programa. Cada variável corresponde a uma posição de memória. Embora uma variável possa assumir valores diferentes, ela só pode armazenar um valor a cada instante.

Declaração de variáveis

Para que se possa usar uma variável em um programa, é necessário fazer uma *declaração de variável* antes.

A declaração de variáveis simplesmente informa ao processador quais são os nomes utilizados para armazenar dados variáveis e quais são os tipos usados. Deste modo o processador pode alocar (reservar) o espaço necessário na memória para a manipulação destas variáveis. É possível declarar mais de uma variável ao mesmo tempo, basta separá-las por vírgulas (,).

A *sintaxe* para declaração de variáveis é a seguinte:

Sintaxe: `tipo variavel_1 [, variavel_2, ...] ;`

em que `tipo` é o tipo de dado e `variavel_1` é o nome da variável a ser declarada. Se houver mais de uma variável do mesmo tipo, seus nomes são separados por vírgulas.

Exemplo:

A declaração de variáveis é feita, em geral, dentro de uma função. Por exemplo, a função principal `main()`. Deste modo se diz que está se fazendo uma declaração de variáveis *locais*. *Variáveis locais* podem ser referenciadas apenas dentro da função dentro da qual foi declarada, neste caso a função `main()`.

É possível fazer a declaração de variáveis fora de uma função. Neste caso diz-se que se fez a declaração de variáveis *globais*.

Iniciação de variáveis

Quando se faz a declaração de uma variável está se determinando que tipo de dado ela vai receber. É possível, em C, declarar uma variável e já armazenar nela um valor inicial. Chamamos este procedimento de inicialização de uma variável.

A *sintaxe* para a iniciação de variáveis é:

Sintaxe: `tipo var_1 = valor_1 [, var_2 = valor_2, ...] ;`

em que `tipo` é o *tipo* de dado, `var_1` é o *nome* da variável a ser inicializada e `valor_1` é o *valor inicial* da variável.

CONVERSÃO DE TIPO (CASTING)

Algumas vezes deseja-se, momentaneamente, modificar o tipo de dado representado por uma variável, isto é, deseja-se que o dado seja apresentado em um tipo diferente do qual a variável foi inicialmente declarada.

Exemplo: Declara-se uma variável como `int` e deseja-se, momentaneamente, que seu conteúdo seja apresentado como `float`. Este procedimento é chamado de *conversão de tipo* ou *casting* (moldagem, em inglês).

A sintaxe da instrução de conversão de tipo é:

Sintaxe: `(tipo) variável`

em que `tipo` é o nome do tipo ao qual deseja-se converter o dado armazenado em `variável`.

CONSTANTES SIMBÓLICAS

- **Constantes definidas pelo programador**

O programador pode definir constantes simbólicas em qualquer programa.

A sintaxe da instrução de definição de uma constante simbólica é:

Sintaxe: `#define nome valor`

em que `#define` é uma diretiva de compilação que diz ao compilador para trocar as ocorrências do texto `nome` por `valor`. Observe que não há **;** (ponto e vírgula) no final da instrução pois trata-se de um comando para o compilador e não para o processador.

A instrução `#define` deve ser escrita antes da instrução de declaração da função principal.

Exemplo:

O uso da diretiva `#define` não se restringe apenas ao apresentado acima, pode-se usá-la para definir *macro instruções*.

- **Constantes pré-definidas**

Em alguns compiladores C, algumas constantes simbólicas já estão pré-definidas. Estas constantes em geral definam alguns valores matemáticos (π, π/2, e, etc.), limites de tipos etc.

Exemplo: A biblioteca `math.h` contém algumas constantes simbólicas pré-definidas.

Constante	Valor	Significado
`M_PI`	3.141593	π
`M_PI_2`	1.570796	π/2
`M_PI_4`	0.785398	π/4
`M_1_PI`	0.318310	1/π
`M_SQRT2`	1.414214	$\sqrt{2}$

Uma biblioteca, em C, é um arquivo pré-compilado chamado arquivo *header* (cabeçalho, em inglês). Em cada biblioteca estão agrupadas constantes e funções semelhantes. Por exemplo, constantes e funções matemáticas estão guardadas na biblioteca `math.h` (*mathematical functions*), constantes e funções de manipulação teclado e monitor estão guardadas na biblioteca conio.h (*console input and output*). Para que se possa usar a constante simbólica em um programa é preciso incluir a biblioteca na compilação do programa.

A sintaxe de inclusão de bibliotecas é a seguinte:

Sintaxe: `#include <nome_bib>`

em que `nome_bib` é o nome da biblioteca que se deseja incluir. Esta instrução deve ser escrita antes do programa principal.

OPERADORES

Um programa tem como característica fundamental a capacidade de processar dados. Processar dados significa realizar operações com estes dados. As operações a serem realizadas com os dados podem ser determinadas por operadores ou funções. Os operadores podem ser de atribuição, aritméticos, de atribuição aritmética, incrementais, relacionais, lógicos e condicionais.

Uma expressão é um arranjo de operadores e operandos. A cada expressão válida é atribuído um valor numérico.

- **Operador de Atribuição**

Um comando de atribuição permite fornecer um valor numérico ou um caractere à uma variável (“guardar um objeto na gaveta”).

Sintaxe: *variável = valor;*
ou
variável = expressão;

Observe que o símbolo de atribuição (=) não tem o mesmo significado que o usual da matemática que representa a igualdade de valores. Este símbolo, em C, representa a atribuição do valor calculado em expressão a variável identificador.

Observe-se também que o operando esquerdo deve ser um identificador de variável, isto é, não pode ser uma constante ou expressão.

Exemplo:

Obs: A ideia de atribuir um valor inicial à variável, ou seja, iniciar a variável com um valor pré-determinado é imprescindível em alguns casos, porque no momento que se cria uma variável o compilador atribui à ela um valor aleatório qualquer (contido na memória do computador).

- **Atribuição múltipla**

E possível atribuir um valor a muitas variáveis em uma única instrução. A esta operação dá-se o nome de atribuição múltipla.

A sintaxe da atribuição múltipla é seguinte:

Sintaxe: `var_1 = [var_2 = ... ] expressão;`

em que `var_1`, `var_2`, ... são os identificadores de variáveis e expressão é uma expressão válida. Observe que na atribuição múltipla as operações ocorrem da direita para a esquerda, isto é, inicialmente o valor de expressão é atribuído a `var_2` e depois o valor de `var_2` é atribuído a `var_1`. Deve-se tomar cuidado com as conversões de tipo e limites de intervalo para atribuições de tipos diferentes.

- **Operadores aritméticos**

Existem cinco operadores aritméticos em C. Cada operador aritmético está relacionado ao uma operação aritmética elementar: *adição*, *subtração*, *multiplicação* e *divisão*. Existe ainda um operador (%) chamado operador de *módulo* cujo significado é o resto da divisão inteira. Os símbolos dos operadores aritméticos são:

OPERADOR	OPERAÇÃO
+	adição
-	subtração
*	multiplicação
/	divisão
%	módulo (resto da divisão inteira)

A sintaxe de uma expressão aritmética é:

Sintaxe: `operando operador operando`

em que `operador` é um dos caracteres mostrados acima e `operando` é uma constante ou um identificador de variável.

- **Restrições de operandos**

Os operandos dos operadores aritméticos devem ser constantes numéricas ou identificadores de variáveis numéricas. Os operadores `+`, `-`, `*`, `/` podem operar números de todos os tipos (inteiros ou reais), porém o operador % somente aceita operandos inteiros.

Exemplo:

Expressão	Valor
`4.8 + 3.5`	
`9 - 3`	
`5.0 * 2.0`	
`10 / 2`	
`37 % 2`	

Uma restrição ao operador de divisão (`/`) é que o denominador *deve* ser diferente de zero. Se alguma operação de divisão por zero for realizada o correra um **erro de execução** do programa (*run-time error*), o programa será abortado e a mensagem `divide error` será exibida.

- **Precedência de operadores**

Quando mais de um operador se encontram em uma expressão aritmética as operações são efetuadas uma de cada vez respeitando algumas regras de precedência: Estas regras de precedência são as mesmas da matemática elementar.

Os operadores de multiplicação (`*`), divisão (`/`) e módulo (`%`) tem precedência sobre os operadores de adição (`+`) e subtração (`-`). Entre operadores de mesma precedência as operações são efetuadas da esquerda para a direita.

Exemplo:

Expressão	Valor	Ordem
`4 + 3 - 2`		
`32 - 4 * 5`		
`4 - 2 * 6 / 4 + 1`		
`6 / 2 + 11 % 3 * 4`		
`6 + 7 % 2 - 2`		

A ordem de precedência dos operadores pode ser quebrada usando-se parênteses: ( ). Os parênteses são, na verdade, operadores de mais alta precedência e são executados primeiro. Parênteses internos são executados primeiro que parênteses externos.

Exemplo:

Expressão	Valor	Ordem
`4 + (3 - 2)`		
`(32 - 4) * 5`		
`(4 - 2 * 6) / 4 + 1`		
`6 / ((2 + 11) % 3) * 4`		
`(6 + 7) % 2 - 2`		

- **Operadores de Atribuição Aritmética**

Muitas vezes deseja-se alterar o valor de uma variável realizando alguma operação aritmética com ela. Como por exemplo: `i = i + 1` ou `val = val * 2`. Embora seja perfeitamente possível escrever estas instruções, foi desenvolvido na linguagem C uma instruções otimizadas com o uso de operadores ditos operadores de atribuição aritmética.

Os símbolos usados são (`+=` , `-=` , `*=` , `/=` , `%=`). Deste modo as instruções acima podem ser rescritas como: `i += 1` e `val *= 2`, respectivamente.

A sintaxe da atribuição aritmética é a seguinte:

Sintaxe:
```
var += exp;
var -= exp;
var *= exp;
var /= exp;
var %= exp;
```

em que `var` é o identificador da variável e `exp` é uma expressão válida. Estas instruções são equivalentes as seguintes:

```
var = var + exp;
var = var - exp;
var = var * exp;
var = var / exp;
var = var % exp;
```

Exemplo:

Atribuição aritmética	Instrução equivalente
`i += 2;`	
`j -= aux;`	
`x *= 2 + y;`	
`num /= 12;`	
`resto %= 3;`	

- **Operadores Incrementais**

Em programação existem instruções muito comuns chamadas de **incremento** e **decremento**. Uma instrução de incremento *adiciona* uma unidade ao conteúdo de uma variável. Uma instrução de decremento *subtrai* uma unidade do conteúdo de uma variável.

Existem, em C, operadores específicos para realizar as operações de incremento (`++`) e decremento (`--`). Eles são genericamente chamados de **operadores incrementais**.

A sintaxe dos operadores incrementais é a seguinte:

Sintaxe:

	instrução equivalente
`++ var`	`var = var + 1`
`var ++`	`var = var + 1`
`-- var`	`var = var - 1`
`var --`	`var = var - 1`

em que `var` é o nome da variável da qual se quer incrementar ou decrementar um unidade.

Observe que existem duas sintaxes possíveis para os operadores: pode-se colocar o operador **à esquerda** ou **á direita** da variável. Nos dois casos o valor da variável será incrementado (ou decrementado) de uma unidade. Porém se o operador for colocado **á esquerda** da variável, o valor da variável será incrementado (ou decrementado) **antes** que a variável seja usada em alguma outra operação. Caso o operador seja colocado **à direita** da variável, o valor da variável será incrementado (ou decrementado) **depois** que a variável for usada em alguma outra operação.

Exemplo:

	Valor das variáveis
`int a, b, c, i = 3;`	
`a = i++;`	
`b = ++i;`	
`c = --i;`	

Os *operadores incrementais* têm a mais alta precedência entre todos, sendo superados apenas pelos parênteses que tem precedência ainda maior.

- **Operadores Relacionais**

Operadores relacionais verificam a relação de magnitude e igualdade entre dois valores. São seis os operadores relacionais em C:

Operador	Significado
`>`	maior que
`<`	menor que
`>=`	maior ou igual a (não menor que)
`<=`	menor ou igual a (não maior que)
`==`	igual a
`!=`	não igual a (diferente de)

A sintaxe das expressões lógicas é:

Sintaxe: `expressão_1 operador expressão_2`

em que `expressão_1` e `expressão_2` são duas expressões numéricas quaisquer, e `operador` é um dos operadores relacionais.

Ao contrário de outras linguagens, em C *não existem* tipos lógicos, portanto o **resultado** de uma **expressão lógica** é um **valor numérico**: uma expressão avaliada **verdadeira** recebe o valor **1**, uma expressão lógica avaliada **falsa** recebe o valor **0**.

Exemplo:
```
int i = 6, j = -2;
float x = 7.3, y = 1.7;
```

Expressão	Valor
`i == 8`	
`x != y`	
`i > x`	
`8 > i`	
`i < j`	
`y <= 3.4`	

Os operadores relacionais de igualdade (== e !=) tem precedência menor que os de magnitude (>, <, >= e <=). Estes, por sua vez, tem precedência menor que os operadores aritméticos. Operadores relacionais de mesma precedência são avaliados da esquerda para a direita.

Exemplo: `int m = 2, n = 3;`

Expressão	Valor	Ordem de Operação
`m + n == 5`		
`m != 2 * n > m`		
`6 >= n < 3 - m`		
`m == n <= m > m`		

- **Operadores Lógicos**

São três os operadores lógicos de C: `&&`, `||` e `!`. Estes operadores tem a mesma significação dos operadores lógicos booleanos AND, OR e NOT.

A sintaxe de uso dos operadores lógicos:

Sintaxe:
```
expr_1 && expr_2
expr_1 || expr_2
!expr
```
em que `expr_1` , `expr_2` e `expr` são expressões quaisquer.

Observe que os operadores lógicos atuam sobre expressões de quaisquer valores. Para estes operadores todo valor numérico diferente de 0 é considerado 1.

Exemplo:

Expressão	Valor lógico
`0`	
`1`	
`1.0`	
`0.8`	

-5.2	
1000	

O resultado da operação lógica `&&` será 1 somente se os dois operandos forem 1, caso contrário o resultado é 0. O resultado da operação lógica `||` será 0 somente se os dois operandos forem 0, caso contrário o resultado é 1. O resultado da operação lógica `!` será 0 se o operandos for 1, e 1 se o operando for 0.

Operador &&

op_1	op_2	op_1 && op_2
1	1	1
1	0	0
0	1	0
0	0	0

Operador ||

op_1	op_2	op_1 || op_2
1	1	1
1	0	1
0	1	1
0	0	0

Operador !

op	!op
1	0
0	1

O Operador `&&` tem precedência sobre o operador `||`. Estes dois têm precedência menor que os operadores relacionais. O operador `!` tem a mesma precedência que os operadores incrementais.

Exemplo: `int a = 0, b = 1, c = 2;`

Expressão	Valor	Ordem de Operação
`a && b`		
`c > b || a < c`		
`a + b && !c - b`		
`!b && c || a`		

- **Operador `sizeof`**

O operador `sizeof` retorna o número de bytes ocupados pelo operando, que pode ser uma variável ou um tipo genérico de dado.

Sintaxe: `sizeof(operando);`

Exemplo:

COMANDOS DE ENTRADA E SAIDA

Através de um teclado, o usuário consegue dar entrada ao programa e aos dados na memória do computador. Por sua vez, o computador pode emitir os resultados e mensagens para o usuário através do vídeo ou da impressora.

COMANDO DE SAÍDA: `função printf()`

A função `printf()` é utilizada para exibir uma mensagem, número, caractere ou qualquer tipo de símbolo no dispositivo padrão de saída de dados, no caso a tela do computador. O protótipo da função `printf()` está contido no arquivo `stdio.h`

Sintaxe: `printf("expressão de controle", argumentos);`

A `expressão de controle` pode ser uma mensagem que o programador deseja imprimir na tela, podendo conter também formatadores padrão que indicam o tipo da variável a ser vinculada nesta mensagem. Os `argumentos` são as próprias variáveis, que podem ser inclusive impressas em forma de operações lógicas ou aritméticas. Cada argumento deve ser separado um d outro por vírgulas (,).

Expressão de controle	
`\n`	*new line* (pula linha)
`\t`	*tab* (tabulação horizontal)
`\b`	*backspace* (volta um caractere)
`\f`	*form feed* (avanço de página)
`\\`	imprimir a barra invertida
`\'`	imprimir o apóstrofe
`\"`	imprimir as aspas

Formatadores	
`%d` ou `%i`	inteiro
`%f`	*float*
`%o`	*octal*
`%x`	*hexadecimal*
`%u`	inteiro sem sinal (*unsigned*)
`%e`	notação científica
`%s`	*string* (cadeia de caracteres)
`%c`	*char*
`%p`	ponteiro
`%ld` ou `%li`	*long int*
`%lf`	*double*

Exemplos:

1)
```
#include <stdio.h>
#include <stdlib.h>
int main () {
  printf ("\nEstou aprendendo a programar em C.\n");
  system ("Pause");
}
```

2)
```
#include <stdio.h>
#include <stdlib.h>
int main () {
  int a = 15, b = 12;
  printf ("\n%d + %d = %d",a,b,a+b);
  system ("Pause");
}
```

3)
```
#include <stdio.h>
#include <stdlib.h>
int main () {
  printf ("\n%d",10<17);  /* 1 */
  system ("Pause");
}
```

4)
```
#include <stdio.h>
#include <stdlib.h>
int main () {
  printf ("\n %d  %c  %x  %o",'a','a','a','a');
  printf ("\n %c  %c  %c  %c\n",'a',97,0x61,0141);
   system ("Pause");
}
```

A tabela ASCII possui 256 caracteres. Se tentar imprimir um número maior que 255 em formato caractere (%c), será impresso o resto da divisão do número por 256.

Exemplo:

```
#include <stdio.h>
#include <stdlib.h>
int main () {
  printf ("\n%c",3393);
  system ("Pause");
}
```

É possível também limitar o número máximo de algarismos significativos nos formatadores.

Exemplos:

1)
```
#include <stdio.h>
int main () {
  float num = 1234.5678;
  printf ("%4.2f\n",num);
  printf ("%3.2f\n",num);
  printf ("%3.1f\n",num);
  printf ("%12.3f\n",num);
}
```

2)
```
#include <stdio.h>
int main () {
  int aux = 42;    float num = 1234.5678;
  printf ("%d\n",aux);
  printf ("%6d\n",aux);
  printf ("%d\n",102);
  printf ("%10d\n",102);
}
```

COMANDO DE LEITURA: `função scanf()`

A função `scanf()` também é uma função de I/O implementada em todos os compiladores C. Ela é o complemento de `printf()` e permite ler dados formatados da entrada padrão (teclado).

Sintaxe: `scanf("expressão de controle", argumentos);`

Os formatadores utilizados na `expressão de controle` são os mesmos utilizados na função `printf()`.A lista de `argumentos` deve consistir nos endereços das variáveis.

A linguagem C oferece um operador para tipos básicos chamado operador de endereço e referenciado pelo símbolo **`&`** que retorna o endereço do operando. A memória do computador é dividida em bytes, e são numerados de 0 até o limite da memória. Estas posições são chamadas de endereços. Toda variável ocupa uma posição na memória, e seu endereço é o primeiro byte ocupado por ela.

Exemplos:

1)

```
#include <stdio.h>
int main () {
  int num;
  printf("Digite um numero: ");
  scanf("%d",&num);
  printf("\nO numero eh %d",num);
  printf("\nO endereco eh %x",&num);
}
```

2)

```
#include <stdio.h>
int main () {
  int dia, mes, ano;
  printf("Digite uma data: ");
  scanf("%d %d %d",&dia,&mes,&ano);
  printf("\nData: %d/%d/%d",dia,mes,ano);
}
```

3)

```
#include <stdio.h>
int main () {
  int dia, mes, ano;
  printf("Digite uma data: ");
  scanf("%d/%d/%d",&dia,&mes,&ano);
  printf("\nData: %d/%d/%d",dia,mes,ano);
}
```

4)

```
#include <stdio.h>
int main() {
  float num, frac;
  int inteiro;
  printf("Digite um numero: ");
  scanf("%f", &num);
  inteiro = num;
  frac = num - inteiro;
  printf("A parte fracionaria de %f e' %f ", num, frac);
}
```

Exercícios

1) Faça um programa na que receba como entrada dois números reais, faça a soma e a média desses dois números e exiba somente a média.

2) Escreva um programa que receba um número n e imprima o valor correspondente ao seu quadrado (n^2).

3) Escreva um programa que receba como entrada o peso e a altura de uma pessoa e calcule o seu IMC. O IMC é dado pela fórmula:

$$IMC = \frac{peso}{altura^2}$$

4) Construa um programa que receba como entrada dois valores inteiros e armazene em uma variável a e b, depois troque os valores de a com b e imprima-os na tela.

5) Escreva um programa que leia uma quantidade em horas, minutos, segundos e informe a quantidade total de segundos equivalente.

6) Construa um programa que leia uma velocidade (v) em m/s e converta para km/h.

7) Faça um programa que leia a quantidade de quilômetros rodados e a quantidade gasta de combustível em litros por um carro e informe a média de combustível gasto em cada quilômetro rodado.

8) Construa um programa que calcule a quantidade de dinheiro gasto por um fumante com cigarros durante n anos. Para isso, é necessário ler a quantidade de cigarros que o fumante fuma por dia, a quantidade de anos que ele fuma e o preço médio de uma carteira de cigarros. (obs: cada carteira de cigarros contém 20 cigarros e, cada ano têm 365 dias.)

9) Faça um programa que receba o valor de uma temperatura em graus Celsius (C) e calcule a sua temperatura correspondente em graus Farenheit (F).

$$C = \frac{(F - 32) * 5}{9}$$

10) Construa um programa que receba o preço de custo de um produto e calcule o preço final do mesmo sabendo:
    - O preço final é calculado através da soma do preço de custo, o valor dos impostos e o lucro esperado.
    - O valor dos impostos é de 45% do valor do preço de custo.
    - O lucro esperado é de 50% do valor do preço de custo.

ALGUMAS FUNÇÕES DE BIBLIOTECA `math.h`

`fabs(x)`	valor absoluto do argumento x
`log(x)`	logaritmo natural do argumento x
`log10(x)`	logaritmo decimal do argumento x
`exp(x)`	e^x (número `e` elevado à potência x)
`sin(x)`	seno do argumento x (em radianos)
`cos(x)`	cosseno do argumento x (em radianos)
`tan(x)`	tangente do argumento x (em radianos)
`sqrt(x)`	raiz quadrada do argumento x
`acos(x)`	arco cujo valor do cosseno é o argumento x
`asin(x)`	arco cujo valor do seno é o argumento x
`atan(x)`	arco cujo valor da tangente é o argumento x
`pow(x,y)`	argumento x elevado ao argumento y
`ceil(x)`	arredonda o número real para cima
`floor(x)`	arredonda o número real para baixo
`round(x)`	arredonda o número real

Exercícios

1) Faça um programa tendo como dados de entrada dois pontos de um plano $P(x_1,y_1)$ e $Q(x_2,y_2)$, imprima a distancia entre eles.

2) Escreva um programa que calcule o volume de uma esfera de raio R, fornecida pelo usuário.

3) Faça um programa que calcule a área de um triangulo, dados os comprimentos de seus lados. A área do triangulo cujos lados tem comprimentos a, b e c é dada por:

$$S = \sqrt{p * (p - a) * (p - b) * (p - c)}$$

em que $p = \frac{a+b+c}{2}$ é o semi-perímetro do triangulo