Fatos sobre C

Concentra-se aqui principais fatos a respeito da linguagem C.

Este capítulo não está completo, continue-o sempre que estiver estudando a linguagem e precisar pegar um fôlego do conteúdo. Paramos [nesta página](https://integrada.minhabiblioteca.com.br/reader/books/9788521632474/epubcfi/6/20%5b%3Bvnd.vst.idref%3Dintro%5d!/4/190/2/4).

História

Embora possua um nome estranho quando comparada com outras linguagens de terceira geração, como FORTRAN, PASCAL ou COBOL, a linguagem C foi criada em **1972** nos Bell Telephone Laboratories por Dennis Ritchie com a finalidade de **permitir a escrita de um sistema operacional** (o **Unix**), utilizando uma linguagem de **relativo alto nível**, **evitando assim o recurso ao Assembly**.

Devido às suas capacidades e através da divulgação do sistema Unix pelas universidades dos Estados Unidos, a linguagem C deixou cedo as portas dos laboratórios Bell, disseminou-se e tornou-se conhecida por todos os tipos de programadores, independentemente dos projetos em que estivessem envolvidos, sendo o livro The C Programming Language, de Kernighan & Ritchie, o único elo comum entre os programadores.

Essa dispersão levou a que diferentes organizações desenvolvessem e utilizassem diferentes versões da linguagem C, criando assim alguns problemas de portabilidade, entre outros. Perante tal estado de coisas, e uma vez que a linguagem C se tinha tornado um verdadeiro fenômeno entre programadores e organizações, **o American National Standards Institute (ANSI) formou em 1983 um comitê para a definição de um padrão da linguagem C**, padrão esse que visa ao funcionamento semelhante de todos os compiladores da linguagem, com especificações muito precisas sobre aquilo que a linguagem deve ou não fazer, seus limites, definições etc.

O nome da linguagem (e a própria linguagem) resulta da evolução de uma outra linguagem de programação, desenvolvida por Ken Thompson também nos Laboratórios Bell, chamada de **B**. Dessa forma, é perfeitamente natural que a evolução da linguagem B desse origem à linguagem C.

Razões da terceira geração

Perante um enorme leque de linguagens de programação disponíveis no mercado, seria necessário que uma delas se destacasse muito em relação às outras para conseguir interessar tantos programadores.

A maior parte das linguagens tem um objetivo a atingir:

* **COBOL** — Processamento de Registros
* **PASCAL** — Ensino das Técnicas de Programação
* **FORTRAN** — Cálculo Científico
* **LISP** e **PROLOG** — Vocacionadas para as áreas de Inteligência Artificial

Razões do C

Quanto à C, a que área de desenvolvimento se destina?

A resposta é — nenhuma em particular. É aquilo que habitualmente se denomina **general purpose**, e esta é uma das suas grandes vantagens, pois **adapta-se ao desenvolvimento de qualquer projeto**, como sistemas operacionais, interfaces gráficas, processamento de registros etc. Por incrível que pareça, **C é também utilizada para escrever os compiladores de outras linguagens**.

C é uma linguagem extremamente potente e flexível.

**Rapidez** — Consegue obter performances **semelhantes às obtidas pelo Assembly, através de instruções de alto nível**, isto é, fazendo uso de instruções semelhantes às utilizadas por linguagens como PASCAL ou COBOL, **mesmo para usar mecanismos de mais baixo nível, como o endereçamento de memória ou a manipulação de bits**.

**Simples** — A sua sintaxe é extremamente simples, e o número de palavras reservadas, de tipos de dados básicos e de operadores é diminuto, reduzindo assim a quantidade de tempo e esforço necessários à aprendizagem da linguagem.

**Portável** — Existe um padrão (ANSI) que define as características de qualquer compilador. Desse modo, o código escrito numa máquina pode ser transportado para outra máquina e compilado sem qualquer alteração (ou com um número reduzido de alterações).

**Popular** — É internacionalmente conhecida e utilizada. Está muito bem documentada em livros, revistas especializadas, manuais etc. Existem compiladores para todo tipo de arquiteturas e computadores.

**Modular** — C permite o desenvolvimento modular de aplicações, facilitando a separação de projetos em módulos distintos e independentes, recorrendo à utilização de funções específicas dentro de cada módulo.

**Alto Nível** — C é considerada uma linguagem de **terceira geração**, tal como PASCAL, COBOL, BASIC etc., as quais são habitualmente denominadas de alto nível quando comparadas com a linguagem Assembly. **C permite, ainda, o acesso à maior parte das funcionalidades de Assembly**, utilizando expressões e instruções de alto nível.

A título de exemplo: é possível manipular a memória diretamente, utilizando o endereço de qualquer objeto (seja variável ou função), atuando diretamente na memória sem qualquer tipo de restrição, o que aumenta a flexibilidade da linguagem.

**Bibliotecas Muito Poderosas** — O fato de C possuir um número reduzido de palavras-chave indica que as capacidades de C são muito limitadas, e na realidade são. A maior parte das funcionalidades da linguagem C é adicionada pela utilização de funções que existem em bibliotecas adicionais e realizam todo tipo de tarefas, desde a escrita de um caractere na tela até o processamento de strings etc.

**Macros** — C tem a possibilidade de utilização de Macros no desenvolvimento de aplicações, reduzindo assim a necessidade de escrita de funções distintas para a realização do mesmo processamento para tipos de dados diferentes. As Macros permitem aumentar a velocidade de execução sem ter que aumentar a complexidade de escrita do código.

**Foco** — A linguagem C permite ao programador escrever o código como bem quiser. Um programa pode ser todo escrito numa única linha ou dividido por inúmeras linhas.

A forma como o código é escrito depende unicamente do gosto do programador. Este tem apenas que se preocupar com o objetivo e a correção da aplicação que está desenvolvendo, de modo a obter no final o resultado desejado. Por isso não é necessário formatar o código a partir da coluna x e os dados a partir da coluna y, como acontece com outras linguagens.

**Evolução** — C é uma linguagem particularmente estável. A evolução das linguagens fez com que também C evoluísse no sentido das Linguagens Orientadas a Objetos, dando origem a uma nova linguagem: C++, a qual mantém a sintaxe da linguagem C e permite um conjunto adicional de características (Encapsulamento, Hereditariedade, Polimorfismo, sobrecarga etc.).

Atualmente, uma nova linguagem — Java — apresenta-se como nova base expandida de trabalho para os programadores. Também essa linguagem se baseia em C e C++.

Filosofia da Programação em C

C é uma linguagem que reflete um pouco a filosofia subjacente ao Unix, incorporando, aliás, algumas das suas características (case sensitive, por exemplo).

O sistema operacional Unix possui um grande conjunto de comandos, sendo cada um deles responsável por realizar uma só tarefa.

Caso se queira realizar uma tarefa complexa, pode-se recorrer a um conjunto de comandos do sistema Unix, encadeando-os através de pipes ou outras estruturas de comunicação, obtendo assim o resultado pretendido.

Dessa forma, a probabilidade de a tarefa ser realizada com deficiências é menor do que se ela fosse resolvida por um único comando, uma vez que a complexidade subjacente a um programa complexo é maior que a soma das complexidades da soma de todos os programas simples que o constituem.

Com tudo isso queremos dizer que é mais fácil implementar ***pequenos pedaços de código que realizem corretamente uma única função, e a realizem bem, do que fazer grandes quantidades de código que utilizem elevado conjunto de condições, variáveis e situações de exceção de forma a atingir o mesmo resultado***.

Assim, um projeto deve ser dividido em módulos.

Introdução

Esta introdução busca formalizar principais tópicos sobre a programação C.

Definição de Programa

Um programa é uma sequência de código organizada de tal forma que permita resolver um determinado problema.

Função principal: main(){}

Em um programa terá que existir um critério que indique ao compilador qual a instrução ou local onde irá começar a execução dele. Desta forma, em C, existe a função “main()”. Para indicar que se trata de uma função, a palavra main é seguida com parênteses, pois em C qualquer função tem que ser seguida por parênteses.

Nela, entre { }, será colocado tudo que queremos que seja executado. Ao conjunto de código existente entre chaves chama-se Bloco.

Case Sensitive

A linguagem C é Case Sensitive, isto é, faz diferenciação entre maiúsculas e minúsculas. Todas as instruções de C são escritas com letra minúscula, e **só se deve utilizar letras maiúsculas quando desejarmos utilizar variáveis, mensagens ou funções escritas por nós**.

Funções externas de Entrada e Saída

A linguagem C **não possui mecanismos de Entrada e Saída**. Para resolver esse problema temos que **recorrer a um conjunto de funções que existem em Bibliotecas** de Funções. Isso implica que temos que adicionar à linguagem um conjunto de outras funcionalidades que, por defeito, ela não nos proporciona.

Definição biblioteca

Para ter acesso a esse conjunto de funções teremos que incluir a sua definição no nosso programa.

Neste caso, o #include <stdio.h>, que não é C, mas sim uma **diretiva que indica ao compilador (mais propriamente ao pré-processador) que deverá adicionar ao processo de compilação um arquivo existente em alguma parte no disco do seu computador**, chamado stdio.h, de forma que o compilador tenha acesso a um conjunto de informações sobre as funções que virá a utilizar.

Esses arquivos têm sempre a **extensão .h, pois não têm código,** mas apenas os cabeçalhos (headers) das funções que representam. São por isso habitualmente designados por **header files**.

Desse modo, a linha #include <stdio.h> significa ''adiciona o arquivo stdio.h ao meu programa''. Exatamente nessa posição. **Como não é uma instrução de C, não é seguida de ;**.

O arquivo stdio.h permite o acesso a todas as funções de entrada e saída normais; stdio quer dizer **standard input/output**.

As linhas começadas por # (#include, #define, etc.) não são C, mas sim diretivas ao pré-processador. Por isso não devem ser seguidas de ponto-e-vírgula.

(Externo) Função de saída: print()

Na realidade, C é uma linguagem com muitas poucas palavras reservadas, desta forma, não é de se surpreender que C não possua mecanismos de Entrada e Saída incorporados. Em vez disso, ela recorre à sua potente biblioteca de funções para fornecer este tipo de serviço.

Uma das funções que permite a escrita na tela é a função printf = print + formatado. Dentro dos parênteses é feita a comunicação com a função. Nesse caso passamos a string (conjunto de caracteres) que queremos que seja escrita — printf("Hello World").

(Externo) Função de saída: puts()

Esta função é também parte da biblioteca de cabeçalhos de funções Standart Input Output. Seu nome significa “put string” e sua diferença em relação à função printf() é que puts() inclui quebra de linha após a string automaticamente.

Desta forma, podemos dizer que puts(“ola”) é equivalente a printf(“ola\n”).

(Externo) Função de entrada: scanf()

Da mesma forma que existe a função printf para a escrita de valores, existe uma função correspondente para a leitura de valores, a função scanf.

#include <stdio.h>

int main(){

int num;

printf(“Introduza um numero: “);

scanf(“%d”,&num);

printf(“O numero introduzido foi %d\n”,num);

}

No caso do programa anterior, queremos ler um valor para uma variável. Para tal usamos a função:

scanf()

O primeiro parâmetro dessa função é uma string com os formatos de leitura. Como só queremos ler uma variável, haverá apenas um formato de leitura. Sendo a variável que queremos ler do tipo inteiro, o formato de leitura será %d.

scanf("%d")

Em seguida, temos que indicar qual a variável que irá receber o valor inteiro a ser lido. Essa variável, como é do tipo inteiro, tem que levar um & antes de seu nome.

scanf("%d", &num);

e assim se obtém a linha 5, que permite realizar a leitura de um inteiro e armazená-lo numa variável.

O inteiro, depois de lido, é guardado na variável num e, em seguida, o seu valor é escrito na tela através da função printf.

O Caractere Especial \

Utilizamos o caractere especial \ em duas ocasiões:

1. Quando queremos usar um símbolo apenas como seu caractere, ou seja, retirar a especialidade de um símbolo.
2. Quando queremos representar um caractere que, de outro modo, seria difícil ou quase impossível de representar.

1

O primeiro caso pode ser verificado ao tentar executar o seguinte código:

printf(“Hoje está um **“**lindo**”** dia.”);

Este código teria um erro de compilação, pois consideraria as aspas que eram para ser simples caracteres como um delimitador.

Para evitar este erro, seria necessário utilizar o caractere especial \ que retiraria sua especialidade e a transformaria num simples caractere.

printf(“Hoje está um \”lindo\” dia.”);

2

Um exemplo que temos utilizado até agora foi o \n, para fazer a quebra de linha.

Lista

Segue abaixo a lista completa das representações que podem ser feitas utilizando o caractere especial \:

|  |  |
| --- | --- |
| **\7** | Bell (sinal sonoro do computador) |
| **\a** | Bell (sinal sonoro do computador) |
| **\b** | BackSpace |
| **\n** | New Line |
| **\r** | Corriage Return |
| **\t** | Tabulação Horizontal |
| **\v** | Tabulação Vertical |
| **\\** | Caractere \ |
| **\’** | Caractere ‘ |
| **\”** | Caractere “ |
| **\?** | Caractere ? |
| **\ooo** | Caractere cujo código ASCII em octal é ooo |
| **\xnn** | Caractere cujo código ASCII em Hexadecimal é nn |
| **%%** | Caractere % |

Comentários

Um comentário em C é qualquer conjunto de caracteres compreendido entre os sinais de /\* e \*/.

Os comentários não se destinam a ser interpretados pelo compilador ou por qualquer componente do processo de desenvolvimento. São **simplesmente ignorados pelo compilador**, e o **programa executável não terá qualquer sinal deles**.

Uma vez que os comentários não têm qualquer interferência num programa, servem apenas para documentação de código. O seu objetivo é facilitar a vida do programador que tem que olhar para um determinado projeto em C, evitando que este tenha que perceber todo o código para saber o que determinado conjunto de instruções faz.

* Mesmo que por grande parte das vezes não recomendável, os comentários podem sempre ser colocados dentro de códigos. Apenas não serão tratados como comentários se você os colocar dentro de uma string.
* Os compiladores não permitem, em geral, a existência de comentários dentro de comentários. Porque seguindo a regra dos comentários em C, o comentário inicial terminava quando encontrasse o símbolo \*/, que encerra o comentário. A extensão do comentário variaria então apenas entre a primeira ocorrência de /\* e a primeira ocorrência de \*/, detectando então um final de comentário sem o correspondente início.

Variáveis

Sempre que desejarmos guardar um valor que, por qualquer razão, não seja fixo, devemos fazê-lo utilizando variáveis.

*- “Uma variável é nada mais que um nome que damos a uma determinada posição de memória para conter um valor de um determinado tipo.”*

Como o seu próprio nome indica, o valor contido em uma variável pode variar ao longo da execução de um programa.

As variáveis são sempre armazenadas em memória, e são uma forma simples de referenciar posições de memória. O tipo que lhes está associado indica o número de bytes que serão utilizados para guardar um valor nessa variável. Texto

Descrição gerada automaticamente

Definição

Uma variável deve ser sempre definida antes de ser usada. A definição de uma variável indica ao compilador qual o tipo de dado que fica atribuído ao nome que indicarmos para essa variável.

A definição de variáveis é feita utilizando a seguinte sintaxe:

tipo nome;

Exemplos:

int num1;

char ch1, novo\_char;

float pi, raio, perímetro;

double total, k123;

Como observado acima, é possível fazer declarações múltiplas de variáveis de mesmo tipo de uma só vez.

*- “A declaração de variáveis tem que ser sempre realizada antes de sua utilização e antes de qualquer instrução.”*

Nomes de Variáveis

O nome que se vai atribuir a variáveis em C implica observar um número reduzido de regras:

* O nome de uma variável pode ser constituído por letras do alfabeto (minúsculas ou maiúsculas), dígitos (0 … 9) e ainda pelo caractere underscore (\_ ).
* **O primeiro caractere não pode ser um dígito**. Terá que ser uma letra ou o caractere underscore. No entanto, é desaconselhável a utilização deste último como primeira letra identificadora de uma variável.
* **Maiúsculas e minúsculas representam caracteres diferentes**, logo variáveis distintas.
* Uma variável **não pode ter por nome uma palavra reservada da própria Linguagem C**. Assim, não podemos ter uma variável denominada float, if ou for, uma vez que essas palavras são instruções ou tipos da própria linguagem.

**Não é aconselhável a utilização de caracteres acentuados** (ã, õ, á, é, etc.) no nome das variáveis, pois a grande maioria dos compiladores não os aceita como caracteres admissíveis.

O caractere underscore ( **\_** ) **é habitualmente utilizado para fazer a separação entre palavras que representam uma única variável**. Ex: Num\_Cliente, Id\_Fatura, Vou\_Continuar etc.

O número de caracteres que o nome de uma variável pode conter depende do compilador, mas é normal que sejam permitidos nomes de variáveis com até **32 caracteres** (ou mais).

### Cuidados a seguir

* O nome de uma variável deve ser descritivo daquilo que ela armazena.
* O nome de uma variável **não deve ser todo escrito em maiúsculas**, pois identificadores totalmente escritos em maiúsculas são tradicionalmente utilizados pelos programadores de C para referenciar **constantes**.
* Caso o nome de uma variável use mais do que uma palavra, utilize o caractere **underscore** **ou** a diferença entre **minúsculas e maiúsculas para as separar**, facilitando assim a leitura.
* Não utilize o caractere *underscore* (\_) para iniciar o nome de uma variável.

Atribuição

Sempre que uma variável é **declarada**, estamos **solicitando** ao **compilador** para **reservar espaço em memória para armazená-la**. Esse espaço passará a ser **referenciado através do nome** da variável.

No caso do inteiro, o **espaço** em bytes que lhe é reservado **varia com as arquiteturas em que é utilizado**. Em microcomputadores o seu valor é normalmente de 2 bytes, enquanto em máquinas maiores é habitualmente de 4 bytes.

Independentemente do número de bytes que ocupe, o nome da variável referencia a totalidade do espaço ocupado pela variável.

Sempre que uma variável é **definida, um conjunto de bytes fica associado a ela**. Ora, esses bytes têm bits com valor 1 e outros bits com 0, constituindo um número qualquer. Dessa forma, quando uma variável é criada fica automaticamente com um valor que não é 0 nem 1, nem qualquer valor predefinido, mas sim um **valor qualquer aleatório que resulta da disposição dos bits** que se encontram nos bytes reservados para a representação dessa variável.



Ao realizar uma atribuição o valor anterior presente na variável é eliminado, ficando nela o novo valor que lhe foi atribuído. A atribuição de um valor só pode ser realizada para variáveis.

Uma atribuição é realizada obedecendo à seguinte sintaxe:

tipo nome = expressão;

Exemplo:

int num = -17;

int num2;

int n1 = 3, n2 = 5;

int a = 10, b, c = -123, d;

Assim como as múltiplas declarações, podemos fazer múltiplas atribuições.

A atribuição de valores em C é realizada através do sinal de **=** , chamado de **sinal de** **atribuição** e lê-se “recebe” ou “tem atribuído” (jamais chame de “igual” ou “igualdade”, pois poderá se confundir. Este símbolo é de atribuição, ele atribui algo a algo, não tem relação nenhuma com igualdade!).

### Ordem de execução de múltiplas atribuições

Observe a seguinte questão:

A = 1;

B = 2;

C = 3;

D = 4;

Qual é o valor das variáveis se, em seguida, fosse executada a seguinte instrução:

A = B = C = D = 5;

A resposta é simples, todas ficam com o valor 5. A razão tem muito a ver com as características da linguagem C. Quando são escritas várias atribuições consecutivas, estas são realizadas não da esquerda para a direita, mas sim da direita para a esquerda.

Diagrama

Descrição gerada automaticamente com confiança baixa



Tipos de Dados Básicos

Estes são apenas quatro (char, int, float e double)1, e serão apresentados detalhadamente em seguida.

Inteiros – int

Uma vez que estamos falando de números inteiros, é possível realizar um conjunto de operações sobre eles, cujo resultado é sempre um valor inteiro.

|  |  |
| --- | --- |
| **Operação** | **Descrição** |
| + | Soma |
| - | Subtração |
| \* | Multiplicação |
| / | Divisão inteira |
| % | Módulo (resto da divisão inteira) |

### Formato

Utilizamos formatos (juntamente a posterior referenciação de qual variável) para devidamente referenciar a posição de valores de variáveis.

*-“O formato de escrita de um inteiro é* ***%d****.”*

Para elucidar, analise a seguinte situação:

#include <stdio.h>

int main(void){

int num1 = 123;

printf(“O valor de num1 é num1”);

}

Veja que num1 está guardado numa variável, e não podemos colocar a variável num dentro da string do printf, uma vez que o printf iria escrever a string “num” em vez do valor que estaria guardado na variável.

Desta forma, para que conseguíssemos devidamente referenciar o valor da variável na posição, precisaríamos utilizar do formato %d e posteriormente colocamos por ordem as variáveis ou os valores que irão ser substituídos em cada %d, separados por vírgula. Da seguinte forma:

#include <stdio.h>

int main(void){

int num1 = 123;

printf(“O valor de num1 é %d”, num1);

}

Variações de espaço

Como foi mencionado anteriormente, o tamanho em bytes de um inteiro varia de arquitetura para arquitetura, sendo os valores mais habituais de 2 ou 4 bytes.

É importante saber qual a dimensão de um inteiro quando se desenvolve uma aplicação, caso contrário corre-se o risco de tentar armazenar um valor numa variável inteira com um nº de bytes insuficiente.

Para saber qual a dimensão de um inteiro (ou de qualquer tipo ou variável), o C disponibiliza um operador denominado sizeof, cuja sintaxe é semelhante à utilizada para invocar uma função.

A sintaxe do operador sizeof é:

**sizeof <expressão> ou sizeof ( <tipo> )**

#include <stdio.h>

main(){

printf("O Tamanho em bytes de um Inteiro = %d\n", sizeof(int));

}

Modificadores de tipo

O fato de o tamanho de um inteiro poder variar é algo preocupante, pois os limites das variáveis que armazenam inteiros podem variar de maneira drástica, reduzindo fortemente a portabilidade dos programas entre máquinas diferentes. Repare bem na diferença entre os valores que uma variável pode conter:

Tabela

Descrição gerada automaticamente

Como podemos então garantir que um programa escrito por nós usa sempre dois ou quatro bytes para armazenar um inteiro, se o tamanho de um inteiro varia de máquina para máquina?

Na declaração de um inteiro podem ser utilizados quatro prefixos distintos para melhor definição das características da variável.

* **short**      — Inteiro pequeno (2 bytes)
* **long**        — Inteiro grande (4 bytes)
* **signed**     — Inteiro com sinal (negativos e positivos)
* **unsigned** — Inteiro sem sinal (apenas positivos)

### Short e long

Para garantirmos que o inteiro n usa **apenas 2 bytes de memória, independentemente da arquitetura** utilizada, devemos declarar a variável como:

**short** int n;

Para garantirmos que o inteiro n usa sempre 4 bytes de memória, independente da arquitetura utilizada, devemos declarar a variável como:

**long** int n;

*- “O formato de leitura e escrita de variáveis inteiras short e long nas funções scanf e printf devem ser precedido dos prefixos h (short) e l (long).”*

scanf(“%**h**d”,&idade);

printf(“Tenho %**l**d na minha conta do banco.”)