Linguagem C

Linguagem C

V1.0.0 (SET/2023)

Sumário

1 Linguagem C

C é uma linguagem de programação de computadores. É possível usá-la para criar um conjunto de instruções para que o computador possa executar. Isso significa que você pode usá-la para criar listas de instruções para um computador seguir. A linguagem C é uma das milhares de linguagens de programação atualmente em uso. Desenvolvida em 1972 por Dennis Ritchie no Bell Lab para uso no sistema operacional Unix, foi amplamente aceita por oferecer aos programadores o máximo em controle e eficiência. Existe há várias décadas e ganhou ampla aceitação por oferecer aos programadores o máximo em controle e eficiência. A linguagem C foi criada com o objetivo principal em mente: facilitar a criação de programas extensos com menos erros, recorrendo ao paradigma da programação algorítmica ou procedimental mas sobrecarregando menos o autor do compilador, cujo trabalho complica-se ao ter de realizar as características complexas da linguagem. Em programação, a linguagem é por onde o *hardware* (máquina) e o programador se comunicam. É um processo formal que funciona por meio de uma série de instruções, símbolos, palavras-chave e regras semânticas.

2 Linguagem C

V1.0.0 (SET/2023)

3 Printf()

3.1 Especificadores de formato: *printf* ( )

Digito	Descrição
d ou *i*	Números inteiros em base decimal
x	Números inteiros em base hexadecimal
f	Números em ponto flutuante (com casas decimais)
e	Números em notação científica (com casas decimais)
c	Caracteres alfanuméricos (texto)
s	Sequencia de caracteres alfanuméricos (texto)
.<num>	Especifica quantos dígitos serão impressos após a virgula

3.2 Sequencia de Escape

Escape	Descrição
\a	Toca um bipe, alarme sonoro padrão do sitema
\b	Backspace
\n	Quebra de linha
\t	Tabulação horizontal (TAB)
\r	Retorna ao início da linha
\o	Caractere nulo
\v	Tabulação vertical

Escape	Descrição
\	Caractere\

Escape	Descrição
\ ’	Caractere /
\ ’ ’	Caractere ’ ’
\ ?	Caractere ?
\ 123	Caractere relacionado ao código 123 em octal (ASCII)
\ X12	Caractere relacionado ao código 12 em hexadecimal (ASCII)
%%	Caractere %

4 Entrada de dados: *scanf( )*

- **Sintaxe:**

```
scanf(<form.>), &<v1>, &<v2>, ..., &<vN>);
```

- **Uso:**
  - 1 ou N variáveis

Digito	Descrição
d ou *i*	Números inteiros em base decimal
x	Números inteiros em base hexadecimal
f	Números em ponto flutuante (com casas decimais)
e	Números em notação científica (com casas decimais)
c	Caracteres alfanuméricos (texto)
s	Sequencia de caracteres alfanuméricos (texto)
`[^chars]`	Lê todos os dados digitados, exceto os especificados em ´´chars´´

OBS: sempre colocar um printf antes do scarnf.

5 Aritmética

5.1 Operadores matemáticos

- **Igualdade:** ==
- **Soma**: +
- **Subtração:** -
- **Multiplicação:** *
- **Divisão:** /
- **Resto de divisão inteira:** %
- **Existe precedência entre operadores!**

5.2 Atribuições aritméticas

- **Operadores de atribuição aritmética**
  - ***Incremento em 1 unidade: ++***
  - ***Decremento em 1 unidade: –***
  - ***Incremento em 1 unidade: +=***
  - ***Decremento genérico: -=***
  - ***Atribuição com multiplicação:*=***
  - ***Atribuição com divisão: /=***

6 Variáveis

- Espaço em memoria para armazenar dados
- *Sintaxe declaração:*

```
1          < tipo >< nome >
```

- *Nomeclatura*

Não pode:
Não pode número em primeira posição Não pode caracteres especiais Não pode espaço

Pode: Letras: “A,B,C…” Underline: ”__” Números depois de nomes: idade2

Atribuição de variáveis

- *Operador:*
  - O que está á direita é atribuido a variavel que esta a direita do operador
- *Sintaxe:*

```
1          < variavel > = < informação >;
```

7 Constantes

- *Diretivas #define*: Cria-se um identificador para um dado de qualquer tipo.
  - Mesmas regras de nomenclatura de variáveis
- **Sintaxe** declaração:

```
1          #define <nome> <valor>
```

8 Comando For

- "‘ Similar ao *while*
  - Inicialização, condição, atualização
- Diferença crucial:
  - Sintaxe mais complexa
  - Tudo fica embutido no comando

OBS: * Muitas vezes os erros não são sintáticos * Debugue se precisar

- ***Sintaxe:***

```
for(<ini.>; <cond.>; <incr.>;){
    <bloco_de_comandos>
}
```

9 Comados If & Else

9.0.1 Estrutura de decisão simples

- **Comando *if*** : Se a condição é verdadeira, o bloco é executado, senão, é ignorado.
- **Sintaxe:**

```
    _if (<condição>) {
        <bloco_de_comandos>
    }_
```

OBS: O comando ***if*** é oque permite que o programa tome as decisões automaticamente. ### Como criar condições lógico-relacionais

Operadores relacionais

- **Maior (>)**
- **Maior ou igual (>=)**
- **Menor (<)**
- **Menor ou igual (<=)**
- **Igual (==)**
- **Diferente (!=)** ##### **Operadores Lógicos**
- **Conjunção (“e” lógico): &&**
  - É verdade quando tudo for verdade
- **Disjunção (“ou” lógico): ||**
  - É verdade se ao menos 1 for verdade
- **Inversão (negação - “não lógico”): !** * É verdade quando o operando é falso

10 Comandos *if-else*:

- **Se a condição for verdadeira**
  - Bloco *if* é executado
  - Bloco *else* é ignorado
- **Caso contrario (condução falsa)**
  - Bloco *if* é ignorado
  - Bloco *else* é executado.
- **Sintaxe:**

```
        _if (<condição>){
            <bloco_de_comandos1>
        }
        else {
            <bloco_de_comandos2>
        }_
```

OBS: Não existe um *else* sem *if*. *else* vem sempre depois do *if*.

```
# Swith Case

* É como se fossem vários _if_ em sequência
* Útil para comparar uma única variável
    * _Somente_ comparações de igualdade
* Caso seja igual a 1 dos valores
    * Executa o respectivo bloco
    * Ignora todos os outros blocos

_OBS:_ Somente para comparações de igualdade.

* **_Sintaxe:_**
``` _switch (<`var>)
```
```

```
 case <`v1`>:
    <bloco_de_comandos>
    breack;
case <`v2`>:
    <bloco_de_comandos2>
    breack;
    ...
case <`vN`>:
    <bloco_de_comandosN>
    breack;
default:
    <bloco_de_comandos_padrão>
    breack;
```

```

# Comandos  While & Do-While

* Inicialização de uma ou mais variáveis de controle.
* Definição de uma condição de parada
    * Enquanto for verdadeira: repete
* Atualização da(s) variável(eis) de controle
* _Cuidado:_ loop infinito

_OBS:_ sempre inicialize a variável.

* **Sintaxe:**
```

```
    while(<condição>){
        <bloco_de_comandos>
    }
```

```
# Comando _do-while_

* Similar ao _while_
    * Inicialização, condição, atualização
* Diferença crucial:
    * Condição avaliada somente ao final
    * Bloco de comandos é executado, obrigatoriamente, ao menos 1x

* **_Sintaxe:_**
```

```
    do{
        <bloco_de_comandos>
    }while(<condição>);
```

```
## Comandos de controle de desvio

* Comando **_break:_**
    * Utilizado para encerrar uma sequência de repetições
* Comando **_continue:_**
    * Utilizado para encerrar a interação atual e passar para a próxima
        .



# Funções

* **Funções:** Resolver problemas complexos através da combinação de
   soluções menores.
* _Sintaxe_ de definição:
```

```
        <tipo> <nome_da_função>(<pârametros>){
            <bloco de comandos>
            return <informação>;
        }
```

```
### Detalhes de uma função

* **Identificador:** mesma regra de variáveis
* Tipo de **retorno**
    * Retorno não é obrigatório em C
* Parâmetros de **entrada**
    * Nenhum, um ou vários

### Parâmetros de função distintos

* **_Sintaxe_** para parâmetros struct:
```

```
    <tipo> <função> (<tipo_struct> <param>){...}
```

```
* **_Sintaxes_** para vetores/matrizes como parâmetro
```

```
    <tipo> <função>(<tipo> <vet>[], int tam) {...}
    <tipo> <função>(<tipo> <vet>[<tam>]) {...}
    <tipo> <função>(<tipo> *<vet>, int tam) {...}
    <tipo> <função>(<tipo m[] [<tam2>], int tam1) {...}
```

```
### Mais detalhes, sobre funções

* **Escopo de variáveis:** _Local vs Global_
* **Declaração de uma função em _C_:**
    * Deve aparecer antes do `main()`
    * _Sintaxe_ de um protótipo de função:
```

```
        <tipo> <nome_da_função>(<parâmetros>);
```

```
# Matrizes

* Matrizes podem ter várias dimensões;
* Dois ou mais índices para acesso a posições;
   * _Sintaxe:_
```

```
        <tipo> <nome>[<dim1>][<dim2>]... [<dimN>];
```

```
### Manipulando uma matriz

* _Sintaxe_ de acesso a posição:
```

```
        <nome>[<i1>][<i2>]... [<iN>]
```

- *Sintaxe* de inicialização:

```
        <declaração> = {{<11>}, {<12>}, ..., {<1N>}};
```

‘‘‘

11 Strings: vetores de char

- Em linguagem C
  - Dados de texto são algo pouco abstrato
  - É preciso entender como os dados estão dispostos em memória
  - Operações sobre caracteres individuais

0	1	2	3	4
O	L	Á	!	\0

OBS: `\0` encerra a string

EXEMPLO: * 10 caracteres = 11 posições * 11 posição = `\0` ## String: entrada e saída

- **Limitações:** Sintaxe rebuscada
- Especificador de formato: `%s`
- *Sintaxe* geral:

```
scanf(´´%s´´, <str>);
```

- *Sintaxe* aprimorada:

```
scanf("%<tam.-1>`` [`^\n]`s", <`str`>);
```

- ***printf( )***
  - Especificador de formato: **%s**
  - *Sintaxe:*

```
printf("<`texto`>", <`str1`>, <`str2`>, ..., <``strN>);
```

11.1 Strings: outras funções de entrada e saída

- Limitações: estouro do limite do vetor
- Sintaxe:

```
gets(<string>);
```

fgets( )

- **Último caractere sempre fica reservado ao** `\0`
- **Entrada padrão:** *stdin*
- Sintaxe:

```
fgets(<string>, <tam>, stdin);
```

puts( )

- Imprime uma string diretamente na tela
- Não admite variáveis de outros tipos
- Sintaxe:

```
puts(<string>);
```

Depois das entradas de dados, chamar sempre *fflush(stdin)* Este comando dá uma limpada no teclado

11.2 Bibliotecca *string.h*

- ***Sintaxe* de funções importantes:**

```
strcpy(<destino>, <origem>);
strcat(<destino>, <origem>);
strlen(<string>);
strcmp(<string1>, <string2>);
```

- Use `strcpy(<destino>, <origem>);` para alterar o conteúdo de uma string. Não use o "="
- Use `strcat(<destino>, <origem>);` para colar uma string na outra
- Use `strlen(<string>);` para mostrar o comprimento da string
- Use `strcmp(<string1>, <string2>);` comparar a igualdade entre strings

11.3 Biblioteca *locale.h*:

```
setlocale(LC_ALL, "Portuguese_Brazil");
```

Esta biblioteca permite que usemos acentos;

12 Structs

- *Sintaxe de definição:*

```
struct <novo_tipo>{
    <tipo1> <campo1>;
    <tipo2> <campo2>;
    ...
    <tipoN> <campoN>;
};
```

12.0.1 Declarando variáveis do novo tipo

- **Comando _typedef:**_ Renomeia um identificador
  - *Sintaxe:*

```
typedef <tipo> <novo_nome>;
```

  - _Sintaxe:

```
struct <novo_tipo> <nome_variável>;
    <novo_nome> <nome_variável>;
```

12.0.2 Acessando os membros de uma _struct

- Antes de mais nada, é preciso haver uma variável desse tipo declarada
  - *Sintaxe:*

```
<variável>.<campo>
```

- **Fato:** É comum misturar vetores e structs!

13 Indo além da função *main( )*

- *main( )*: Seu programa é apenas uma função!
  - Parâmetros de um programa:
    - _int argc
    - _char *argv[ ]_
  - ***Sintaxe*** correta do *main ( )*

```
1           int main(int argc, char *argv[]){...}
```

14 Vetores

- Estruturas de dados uni-dimencionais
- Índice único controla as posições
- ***Sintaxe*** de declaração:

```
1               <tipo> <nome> <tamanh
```

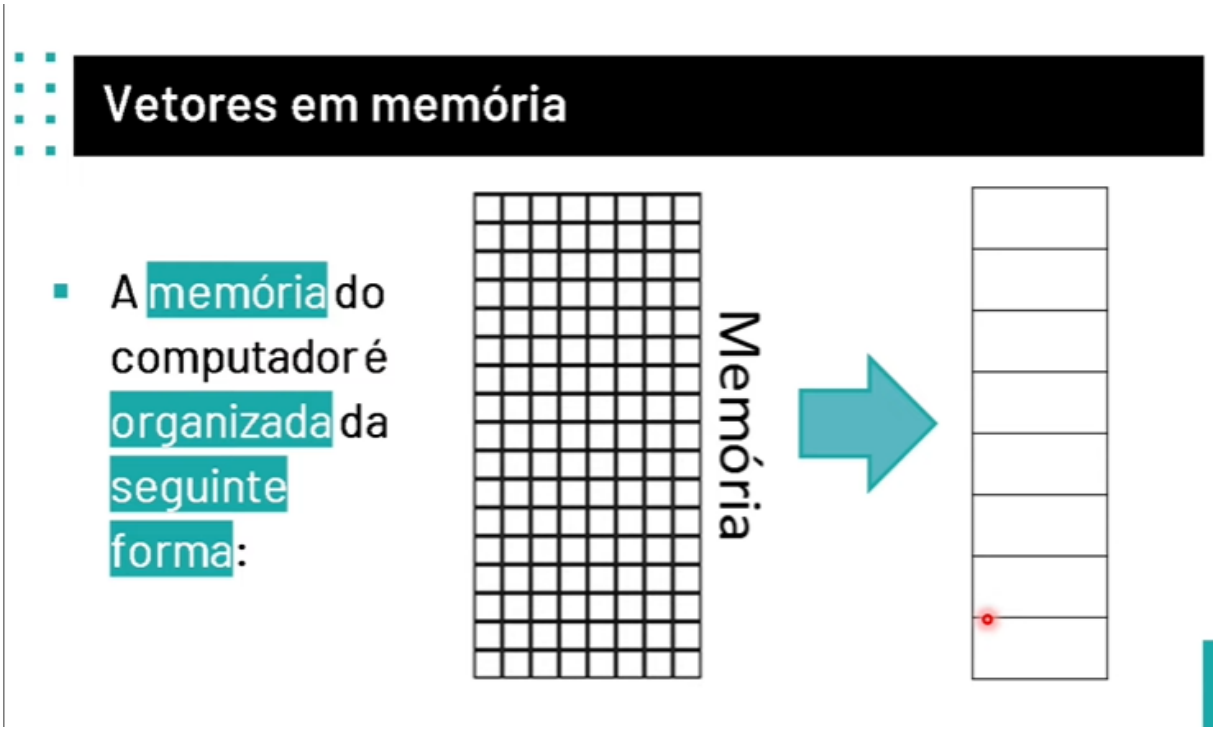

Figura 1: imagem

14.0.1 Manipulando Vetores

- Acesso a uma posição:
  - Não é possível acessar um vetor todo
- ***Sintaxe***

```
1               <nome>[<índice>]
```

- **Lista de inicialização:** Preenche um vetor todo, de uma vez só

```
1          <tipo> <nome>[<tam.>] = {<v1>, <v2>, <v3>, ..., <vN>};
```

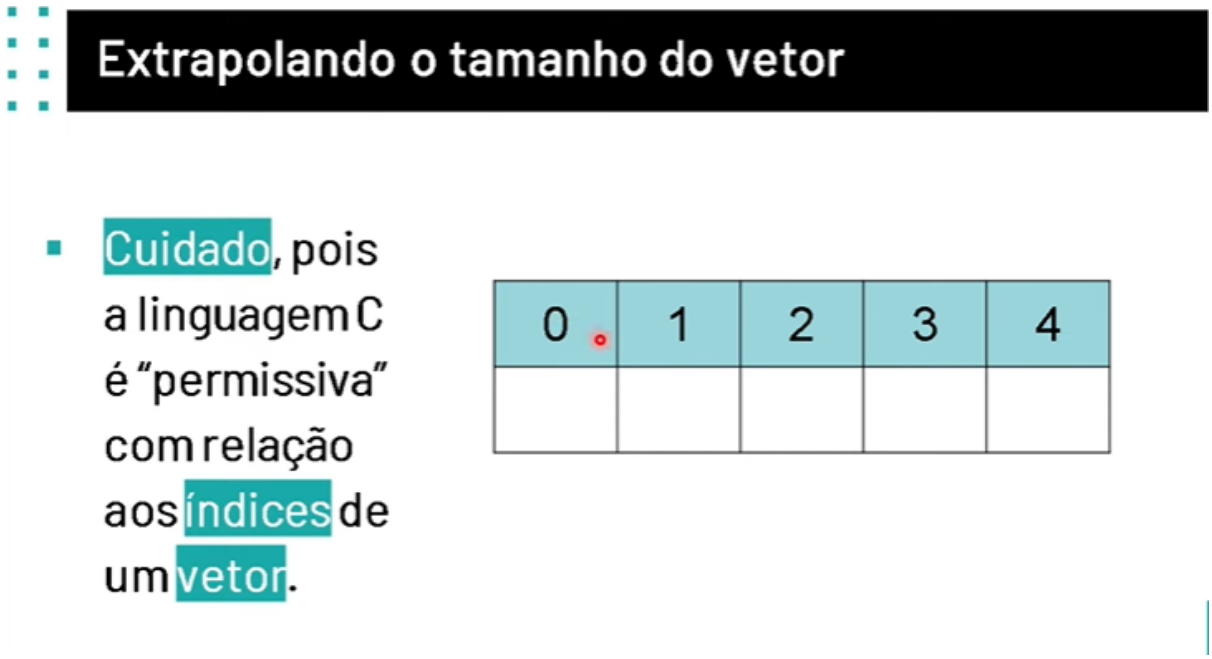

Figura 2: imagem

15 Change log

15.1 Versão 1.0.0

- está é a primeira versão.