
Comandos Linguagem C

Comandos para aprendizagem da Linguagem C

V1.0.0 (OCT/2023)



Sumário

0.1	Especificadores do formato Printf	3
0.2	Operadores aritméticos	4
0.3	Comando IF	4
0.4	Comando IF-ELSE	5
0.5	Comandos lógicos-relacionais	5
0.6	Switch-case	5
0.7	Comando While	6
0.8	Comando Do-While	6
0.9	Comando For	7
0.10	Comando Break	7
0.11	Comando Continue	8
0.12	Vetores	8
0.13	Matrizes	10
0.14	Structs	11
0.15	Comando System CLS	12
0.16	Comando de função de definição	12
1	Change log	13
1.1	Version 1.0.0	13

Printf ("texto", v1, v2, . . . vn);

Printf - É o código para escrever textos Texto - Texto que vai ser mostrado \n - Quebra de linha

Scanf (<form.>), &<v1>, &<v2> . . . , &<vN>); <Form.> - Serve para especificar o formato da variável
Tudo que for adicionado no ao "<form.>", vai estar diretamente ligada a variável Variável tem que
ter sempre um "&" antes da variável. ## Variável regras

Não pode começar com número Não pode caracteres especiais Não pode espaço

Char usado para caractere, necessita especificar o máximo de caractere, senão especificado será armazenado apenas 1 caractere (letra). Ex: **char** nome [50] = "";

int usado para número inteiro Ex: **int**idade = 0;

Float usado para números com casas decimais Ex: **Float**altura = 0,0;

Double usado para várias casas decimais

Variável recebe a informação **Variável** = **Informação**;

Constante **define** <nome> <valor> Toda vez que o <nome> for citado irá ser trocado por <valor>.

0.1 Especificadores do 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	Sequência de caracteres alfanuméricos (texto)
.<num>	Especifica quantos dígitos serão impressos após a virgula

Sequências de escape

Escape	Descrição
\a	toca um bipe, alarme sonoro padrão do sistema
\b	Backspace
\n	Quebra de linha
\t	Tabulação horizontal
\r	Retorna ao início da linha
\0	Caractere nulo
\v	Tabulação vertical
\\	Caractere \
\'	Caractere '
\	Caractere "
\?	Caractere ?
\123	Caractere relacionado ao código 123 em octal (ASCII)
\x12	Caractere relacionado ao código 12 em hexadecimal(ASCII)
%%	Caractere %

0.2 Operadores aritméticos

Existe precedência de operadores

Nome	Sinal
adição	+
Subtração	-
Multiplicação	*
Divisão	/
Resto divisão inteira	%

SINTAXE Variável (soma) = variável (x) + variável (y) soma = A + B; subtr = A - B; multi = A * B; divis = A / B;

Para mostrar na tela deve se usar: **printf** ("%d.", &Variável soma) **%d** se for número inteiro.

%f se houver casas decimais, se necessário informar a quantidade de casas decimais **printf** ("%2f", &Variável soma); **%2f** para mostrar apenas duas casas decimais

Operadores relacionais

Maior que	>
Maior ou igual	>=
Menor que	<
Menor ou igual	<=
Igual	==
Diferente	!=

0.3 Comando IF

O comando **if** é um definidor de bloco, logo não poderá usar ;

SINTAXE

```
1 int main (){
2     int ~~
3     printf ~~
4     scanf ~~
5     **if** (variável >= variável){
6         printf("aprovado!");
7     }
8 }
```

Se estiver na condição verdadeira o **if** irá agir **executando: printf("aprovado!");** Caso a condição for falsa o código será ignorado e pulará para a próxima linha/bloco.

0.4 Comando IF-ELSE

SINTAXE DE IF-ELSE EM CONJUNTO O comando **else** também é definidor de bloco, logo não poderá usar ;

```
1 int main (){
2     int ~~
3     printf ~~
4     scanf ~~
5     **if** (variável >= variável){
6         printf("aprovado!");
7     }
8     else
9         {printf("Reprovado!");
10    }
11 }
```

Se o comando **if** for ignorado o comando **else** será executado. Caso o **if** seja executado, o **else** será ignorado. Não existe **else** sem **if**

0.5 Comandos lógicos-relacionais

Comandos condicionais: Conjunção("e" lógico): && * A expressão vai ser verdadeira se tudo for verdadeiro. **Disjunção**("ou" lógico): || * A expressão vai ser verdadeira se ao menos 1 for verdadeiro. **Inversão**(inversão do valor lógico): ! * É verdade quando o operando for falso.

0.6 Switch-case

É utilizado para compara igualdades e para criar "menus" **Somente para comparações de igualdade (não serve para <, >, <=, >=)** Comparações de única variável

SINTAXE Switch-case

```
1  switch(<var>){
2      caso (var1):
3          <bloco de comandos>
4          break;
5      caso (var2):
6          <bloco de comandos>
7          break;
8      caso (var3):
9          <bloco de comandos>
10         break;
11     caso (varN):
12         <bloco de comandos>
13         break;
14     default:
15         <bloco de comandos>
16         break;
17 }
```

Switch case irá selecionar o caso/opção específica e executar, os restantes dos casos/opções será ignorado. Se nenhum dos casos for escolhidos, então será iniciado o default (não é obrigatório adicionar um default)

0.7 Comando While

Inicialização de 1 ou mais variáveis de controle. Definição de uma condição de parada. Enquanto for verdade irá repetir.

SINTAXE While

```
1  int main(){
2      int i=1
3      while(i<=10){
4          (printf ("%d", i);
5          i++;
6      }
7  }
```

0.8 Comando Do-While

Parecido com While, porém o comando **do** faz com que um bloco de comando seja iniciado/testado antes de começar a repetição, no caso o **do** faz com que seja testado obrigatoriamente na primeira vez

SINTAXE Do-While

```
1 Do{
2     <bloco de comandos>
3 }
4 while(condição);
```

Obs: ao usar o comando do-while, no fim o while receberá o ; pois quem está possuindo o bloco de comando é o comando do

0.9 Comando For

Parecido com While Inicialização, condição e atualização diferença crucial sintaxe mais complexa e tudo fica dentro do comando

SINTAXE For

```
1 for (<INICIALIZAÇÃO>; <CONDIÇÃO>; <INCREMENTO>){
2     <bloco de comando>
```

SEQUÊNCIA DA SINTAXE For

```
1 1º VEZ
2 INICIALIZAÇÃO > CONDIÇÃO VERDADE > BLOCO DE COMANDOS
3 2º VEZ
4 INCREMENTO > CONDIÇÃO VERDADE > BLOCO DE COMANDOS
5 3º VEZ
6 INCREMENTO > CONDIÇÃO VERDADE > BLOCO DE COMANDOS
7 4º VEZ
8 INCREMENTO > CONDIÇÃO VERDADE > BLOCO DE COMANDOS
9 ...
10 nº VEZ
11 INCREMENTO > CONDIÇÃO FALSO > FIM DE REPETIÇÃO.
```

0.10 Comando Break

Comando break interrompe a ação, dentro do laço de repetição, tudo que vier depois do break será ignorado.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  in t main(){
4  int 1;
5  for (i=1; i<=10; i++){
6      if (1 == 5){
7          break;
8      }
9      printf("%d", i);
10 }
```

0.11 Comando Continue

Comando continue, ele faz com que a ação seja ignorada e salta para próxima interação

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main(){
4  int 1;
5  for (i=1; i<=10; i++){
6      if (1 == 5){
7          continue;
8      }
9      printf("%d", i);
10 }
```

0.12 Vetores

Estrutura de dados unidimensionais Índice único controla as posições Sintaxe de declaração:

```
1  tipo, nome, [tamanho];
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  int main(){
6  int v[5] = {10, 20, 30, 40, 50};
7  int i;
8  float s=0;
9      for(i=0; i<5; i++){
10         s += v [i];}
11  printf("Resultado: %f", s/5);
12 }
```

Pedindo para usuário inserir o vetor.


```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int main(){
4     int i;
5     for (i=0;i<5;i++)
6         printf("Insira um dado");
7         scanf("%.0f", v[i]);
8 }
```

Saída Limitações: sintaxe rebuscada Especificador de formato: **%s** Não é necessário usar o **&** no **scanf** para identificar o vetor. Sempre que usar entrada de dados é interessante utilizar o comando **fflush** para não ocorrer problemas com entradas especificamente do tipo **float** e **char** Sintaxe fflush: fflush(stdin); Sintaxe geral:

```
1 scanf("%s", str);
```

Com essa sintaxe geral, o **scanf** impossibilita o usuário de utilizar espaço, se acontecer o **scanf** irá ignorar tudo após o espaço.

Se o usuário escrever mais dígitos que a capacidade que a string suporta o **scanf** irá permitir a leitura acontecer, fazendo com que seu código dê algum erro.

Sintaxe aprimorada:

```
1 scanf("%tamanho-1[``^\\n``]s", str);
```

Tamanho -1 pois não podemos usar o tamanho total, pois o último do caractere é para o **\0** que indica o fim do que foi escrito e é colocado automaticamente pelo **scanf**

Na sintaxe aprimora você é capaz de: Adicionar limite de dígitos que o usuário pode escrever **[^\\n]** Faz com que o **scanf** leia tudo que o usuário escreveu antes do **enter**

Entrada Especificador de formato: **%s** Sintaxe:

```
1 Printf("texto", str1, str2, str3, str4, str5, ..., strN);
```

0.12.1 Outras funções de entrada e saída

gets() Limitações: estouro do limite do vetor Sintaxe:

```
1 gets(string);
```

fgets Último caractere sempre fica reservado para **\0** Entrada padrão: stdin Sintaxe:

```
1 fgets(string, tamanho, stdin);
```

puts

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

```
1 puts(string);
```

fflush(stdin): chamar sempre após uma entrada, para evitar problemas com entradas específicas:
float e **char** Basicamente esse comando serve para limpar lixo armazenado fazendo com que o próximo **scanf** seja utilizado perfeitamente.

0.12.2 Biblioteca de string.h

Sintaxe de funções importantes:

```
1 strcpy (destino, origem);
```

strcpy serve para atribuir/modificar alguma coisa em uma string via código

```
1 strcat (destino, origem);
```

strcat serve para “colar” uma string na outra

```
1 strlen (string);
```

strlen mostra o tamanho da variável string

```
1 strlen (string1, string2);
```

strcmp compara se uma string é igual a outra, se sim irá produzir o valor 0, senão irá produzir um valor diferente de 0.

Biblioteca locale.h

```
1 setlocale (LC_ALL, "portuguese");
```

Faz com que seja possível a utilização de acentuações.

0.13 Matrizes

Matrizes podem ter várias dimensões Dois ou mais índices para acesso a posições **Índice é na vertical e Coluna é na horizontal** primeiro vem índice depois a coluna. Sintaxe:

```
1 <Tipo> <nome> [<dim1>][<dim2>][<dim3>]...[dimN];
```

Precisa dar um tamanho para cada dimensão, pode ser tamanhos diferentes ou iguais.

Manipulando matrizes Sintaxe de acesso a posição:

```
1 Nome [i1][i2]...[iN]
```

Sintaxe de inicialização:

```
1 Declaração = {{i1}, {i2}, ..., {iN}};
```

0.14 Structs

Definido de novo tipo Comando typedef Declarando variáveis do novo tipo Acessando membros de uma variável **structs**

Structs(registros) Sintaxe **Struct**

```
1 struct novo_tipo{
2     tipo1 campo1;
3     tipo2 campo2;
4     tipo3 campo3;
5     ...
6     tipoN campoN;
7 };
```

Regra para o nome do novo tipo é as mesmas regras para identificadores

Comando **typedef** serve para renomear o **struct**

Sintaxe **typedef**

```
1 Typedef tipo novo_nome;
```

Sintaxe de declaração de variável struct

```
1 struct novo_tipo nome_variável;
2     novo_nome nome_variável;
```

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

```
1 Variável.campo
```

Fato: É comum misturar vetores e structs

0.15 Comando System CLS

Código que serve para limpar a tela de operação

Sintaxe System CLS

```
1 System ("cls")
```

0.16 Comando de função de definição

Serve para resolver problemas afim de diminuir a complexidade do código com combinações menores

Sintaxe de Função de definição

```
1 tipo nome_da_função (parâmetros){  
2     bloco_de_comandos  
3     return informação;  
4 }
```

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

Sintaxe para Parâmetros de Função

```
1 Tipo função (tipo_struct parâmetro){...}
```

Sintaxe para Vetores/Matrizes como Parâmetro

Para Vetores (3 distintas)

```
1 tipo nome_função (tipo vetor [], int tamanho){Manipulando o vetor}  
2 tipo nome_função (tipo vetor [tamanho]){...}  
3 tipo nome_função (tipo *vetor, int tamanho){...}
```

Para Matriz

```
1 tipo nome_função (tipo m[][tamanho2], int tamanho1){...}
```

1 Change log

1.1 Version 1.0.0

- Documento para aprendizagem da linguagem C