Comandos Linguagem C

Comandos para aprendizagem da Linguagem C

V1.0.0 (OCT/2023)

Sumário

1 Sintaxe

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

`Variavel` = `Informacao`;

Constante

`define <nome> <valor>`

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

2 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 inicio 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 %

Para Scanf:

Digito	descrição
d ou i	Números inteiros em base decimal
x	Números inteiros em base hexadecimal
f	Números inteiros em ponto flutuante (com casas decimais)
e	Números em notação científica (com casas decimais)
c	Caracteres alfanuméricos (texto)
s	Sequência de caracteres alfanúmericos(texto)
[^chars]	lê os dados digitados abaixo, exceto os especificos em “chars”

3 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	<=

Maior que	>
Igual	==
Diferente	!=

Atribuições aritméticas

Incremento em 1 unidade	++
Decremento em 1 unidade	–
Incremento genérico	+=
Decremento genérico	-=
Atribuição com multiplicação	*=
Atribuição com divisão	/=

4 Comando IF

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

SINTAXE

```
int main (){
    int ~~
    printf ~~
    scanf ~~
        if (variavel >= variavel){
        printf("aprovado!");
        }
}
```

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.

4.1 Comando IF-ELSE

SINTAXE DE IF-ELSE EM CONJUNTO

O comando **else** também é definidor de bloco, logo não poderá usar **;**

```
int main (){
    int ~~
    printf ~~
    scanf ~~
        if (variavel >= variavel){
            printf("aprovado!");
            }
        else
            {printf("Reprovado!");
        }
}
```

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**

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.

5.1 Switch-case

É utilizado para comparar igualdades e para criar “menus”

Somente para comparações de igualdade (não serve para <, >, <=, >=)

Comparações de única variável

SINTAXE Switch-case

```
switch(<var>){
    caso (var1):
        <bloco de comandos>
        break;
    caso (var2):
        <bloco de comandos>
        break;
    caso (var3):
        <bloco de comandos>
        break;
    caso (varN):
        <bloco de comandos>
        break;
    default:
        <bloco de comandos>
        break;
}
```

Switch case irá selecionar o caso/opção especifica 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)

6 Comando While

Inicialização de 1 ou mais variáveis de controle.

Definição de uma condição de parada.

Enquanto for verdade irá repetir.

SINTAXE While

```
int main(){
    int i=1;
        while(i<=10){
            (printf ("%d", i);
            i++;
        }
}
```

6.1 Comando Do-While

Parecido com While, porem 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

```
Do{
    <bloco de comandos>
}
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.

7 Comando FOR

Parecido com While

Inicialização, condição e atualização

diferença crucial

sintaxe mais complexa e tudo fica dentro do comando

SINTAXE FOR

```
for (<INICIALIZAÇÃO>; <CONDICAO>; <INCREMENTO>){
    <bloco de comando>
}
```

SEQUÊNCIA DA SINTAXE FOR

```
1º VEZ
INICIALIZAÇÃO > CONDICAO VERDADE > BLOCO DE COMANDOS
2º VEZ
INCREMENTO > CONDICAO VERDADE > BLOCO DE COMANDOS
3º VEZ
INCREMENTO > CONDICAO VERDADE > BLOCO DE COMANDOS
4º VEZ
INCREMENTO > CONDICAO VERDADE > BLOCO DE COMANDOS
...
nº VEZ
INCREMENTO > CONDICAO FALSO > FIM DE REPETIÇÃO.
```

8 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    }
```

8.1 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    }
```

9 Vetores

Estrutura de dados unidimensionais

Índice único controla as posições

Sintaxe de declaração:

```
tipo, nome, [tamanho];

#include <stdio.h>
#include <stdlib.h>
int main(){
int v[5] = {10, 20, 30, 40, 50};
int i;
float s=0;
    for(i=0; i<5; i++){
    s += v [i];}
printf("Resultado: %f", s/5);
}
```

Pedindo para usuário inserir o vetor.

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

9.1 Saídas

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 ffluxe: fflush(stdin); Sintaxe geral:

```
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 índica 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**

9.2 Entradas

Especificador de formato: %s

Sintaxe:

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

10 Outras funções de entrada e saída

gets( )

Limitações: estouro do limite do vetor

Sintaxe:

```
gets(string);
```

fgets

Último caractere sempre fica reservado para `\0`

Entrada padrão: stdin

Sintaxe:

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

puts

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

Sintaxe:

```
puts(string);
```

fflush(stdin): chamar sempre após uma entrada, para evitar problemas com entradas especificas: **float** e **char**

Basicamente esse comando serve para limpar lixos armazenado fazendo com que o próximo **scanf** seja utilizado perfeitamente.

10.0.1 Biblioteca de string.h

Sintaxe de funções importantes:

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

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

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

strcat serve para “colar” uma string na outra

```
strlen (string);
```

strlen mostra o tamanho da variável string

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

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

Biblioteca locale.h

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

Faz com que seja possível a utilização de acentuações.

11 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:

```
<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:

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

Sintaxe de inicialização:

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

12 Structs

Definidor de novo tipo

```
Comando typedef
```

Declarando variáveis do novo tipo

Acessando membros de uma variável **structs**

Structs(registros)

Sintaxe **Struct**

```
struct novo_tipo{
    tipo1 campo1;
    tipo2 campo2;
    tipo3 campo3;
    ...
    tipoN campoN;
    };
```

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

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

Sintaxe **typedef**:

```
Typedef tipo novo_nome;
```

Sintaxe de declaração de variável struct:

```
struct novo_tipo nome_variavel;
        novo_nome nome_variavel;
```

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

Sintaxe:

```
Variavel.campo
```

É comum misturar vetores e structs

13 Comando System CLS

Código que serve para limpar a tela de operação.

Sintaxe System CLS

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

13.1 Comando de função de dfinição

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

Sintaxe de Função de dfinição

```
tipo nome_da_função (parametros){
    bloco_de_comandos
    return informação;
    }
```

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

```
Tipo função (tipo_struct parametro){...}
```

Sintaxe para Vetores/Matrizes como Parâmetro

Para Vetores (3 distintas)

```
tipo nome_funcao (tipo vetor [], int tamanho){Manipulando o vetor}
tipo nome_funcao (tipo vetor [tamanho]){...}
tipo nome_funcao (tipo *vetor, int tamanho){...}
```

Para Matriz

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

14 Change log

14.1 Version 1.0.0

- Documento para apredizagem da linguagem C