

# Algoritmo e Programação

---



# Algoritmos x Programação

---

**Algoritmo** é um conjunto finito de regras, bem definidas, para a solução de um problema em um tempo finito.

**Programa** é um algoritmo codificado (escrito) em uma linguagem de programação (C/C++).

A **linguagem C** foi criada por Dennis Ritchie, em 1972, no centro de Pesquisas da Bell Laboratories. Ela é uma linguagem de propósito geral, sendo adequada à programação estruturada.


# Diretivas de Compilação

---

Comandos que indicam algumas tarefas a serem executadas antes do programa, como por exemplo a inclusão de uma biblioteca.

Biblioteca é um conjunto de comandos já criados pela linguagem e que devem ser carregados previamente para poderem ser utilizados no programa.

Sempre começa com **#include** e os arquivos tem extensão **.h**.

Ex: `#include <stdio.h>`  inclui os comandos de entrada e saída de dados

# Sintaxe do Programa

---

Todo programa C inicia sua execução na função **main()**. No início da função existe um **{** e no final um **}**.

Cada instrução encerra com **;** (ponto e vírgula) que faz parte do comando.

Comentários são informações que não serão executadas. Elas devem ser colocados das seguintes formas:

**/\* comentário \*/** ou  
**// comentário**

# Sintaxe do Programa

---

A estrutura básica de um programa C é:

```
#include <stdio.h>
```

```
main( )
```

```
{
```

```
    // Área de declaração de variáveis
```

```
    /* Área de comandos do programa */
```

```
}
```

# Constantes e Variáveis

---

O nome das variáveis deve sempre começar com uma letra ou com o caractere '\_' e pode ser seguido por um ou mais caracteres alfanuméricos.

A linguagem C é "*case sensitive*", ou seja, para ela a variável X e a variável x são diferentes.

Todas as variáveis devem ser declaradas antes de poderem ser utilizadas. A forma de declarar é:

**tipo nome\_variável;**

# Tipos de Dados

---

Os tipos de dados existentes na linguagem C são:

- **int** : número inteiro
- **float** : número decimal
- **char** : caracter

Comparando com algoritmo temos:

inteiro	→	int
real	→	float
caracter	→	char

# Atribuição de Valores

---

Para atribuir um valor a uma variável na linguagem C utiliza-se o sinal de igual (=).

Ex:

```
int a;    //cria a variável de nome a
a = 0;    //atribui o valor 0 na variável a
```

A atribuição pode ser feita no momento da declaração da variável:

```
int a = 0;
```

Comparando com algoritmo temos:

a <- 0          a = 0;



# Operadores Aritméticos

---

Soma:	+
Subtração:	-
Multiplicação:	*
Divisão:	/
Resto da Divisão:	%

Na linguagem C, as operações podem ter seus operadores de forma combinada:

<code>x=x+1;</code>	$\longrightarrow$	<code>x+=1;</code>
<code>x=x-5;</code>	$\longrightarrow$	<code>x-=5;</code>
<code>x=x*(y+1);</code>	$\longrightarrow$	<code>x*=y+1;</code>

# Operadores Relacionais

---

Maior que: >

Maior ou igual: >=

Menor que: <

Menor ou igual: <=

Igual a: ==

Diferente: !=

Os resultados desse operadores sempre são Verdadeiro (número diferente de zero) ou Falso (número igual a zero).

# Incremento e Decremento

---

Na linguagem C existe uma forma direta de incrementar o valor de uma variável, ou seja, de somar 1 ao seu valor:

**++x;** //incrementa x antes de usar seu valor

**x++;** //incrementa x depois de ter utilizado seu valor

Ex:

```
#include <stdio.h>
main( )
{
    int x=10;
    printf ("x=%d\n",x++);           //imprime o valor 10
    printf ("x=%d\n",x);             //imprime o valor 11
    printf ("x=%d\n",++x);           //imprime o valor 12
    printf ("x=%d\n",x);             //imprime o valor 12
}
```

# Saída de Dados

---

A função padrão da linguagem C para imprimir mensagens e valores na tela é o **printf()**.

```
printf("expressão", argumentos);
```

Comparando com algoritmo temos:

```
escreva("expressão", argumentos)
```



```
printf("expressão", argumentos);
```

# Saída de Dados

---

Para imprimir valor de argumento deve-se incluir na expressão um controle que indica o formato do valor que será exibido. Ex:

```
printf("Cinco é igual a %d", 5);
```

Os controles que indicam os formatos são:

<b>%d</b>	→	inteiro
<b>%f</b>	→	float
<b>%c</b>	→	char

Podem existir vários argumentos no mesmo comando, basta separá-los com uma vírgula.

# Saída de Dados

---

Na expressão podem ser incluídos caracteres de controle como:

**\n** → nova linha

**\t** → tab

**\b** → volta um caractere

**\"** → mostra caractere aspas

**\\** → mostra caractere barra

# Saída de Dados

---

Para determinar a quantidade de casas decimais que será usada ao imprimir um número real deve-se modificar o controle acrescentando essa informação. Ex:

```
#include <stdio.h>
main()
{
    float x=1234,56;
    printf("%f", x);           //imprime 1234,56000000
    printf("%4:2f",x);        //imprime 1234,56
    printf("%10:1f",x);       //imprime      1234,5
}
```

# Entrada de Dados

---

A função padrão da linguagem C para entrada de dados pelo teclado é o **scanf()**.

```
scanf("expressão", argumentos);
```

Comparando com algoritmo temos:

```
leia(variável)
```



```
scanf("expressão", argumentos);
```



# Entrada de Dados

---

Na expressão deve-se determinar o tipo de valor que será lido (%d ou %f ou %c) e no argumento tem o nome da variável que receberá o valor lido precedido pelo **&** para indicar que é endereço de memória. Ex:

```
scanf("%d", &x); //lê um valor inteiro para variável x
```

```
scanf("%f", &y); //lê um valor real para variável y
```

```
scanf("%c", &z); //lê um caractere para variável z
```

# Comandos Condicionais

---

Na linguagem C o comando condicional mais simples é o **if – else**.

```
if (condição)
    comando1;
else
    comando2;
```

Comparando com algoritmo temos:

```
se condição entao
    comando1
Senao
    comando2
fimse
```



```
if (condição)
    comando1;
else
    comando2;
```

# Comandos Condicionais

---

Ex:

```
#include<stdio.h>
main()
{
    int a,b;
    printf("digite dois números:");
    scanf("%d%d",&a,&b);
    if (b!=0)
        printf("%d\n",a/b);
    else
        printf("divisão por zero\n");
}
```

# Comandos Condicionais

---

No exemplo anterior o comando if tinha apenas uma única instrução a ser executada. Para que possam ser executadas várias instruções é necessária usar a representação de instrução composta :

- Uma chave aberta ( { )
- Uma sequência de instruções (cada uma terminada por ponto e vírgula ;)
- Uma chave fechada ( } )

# Comandos Condicionais

---

Existe ainda uma estrutura condicional para casos de seleção múltipla: o **switch - case**.

```
switch(variável)
{
    case valor1:
        comandos 1;
        break;
    case valor2:
        comandos 2;
        break;
    ...
    default:
        comandos;
}
```



# Comandos Condicionais

---

A variável é testada sucessivamente comparando com os valores de cada caso. Ao encontrar uma coincidência, o comando ou o bloco de comandos é executado.

Se nenhuma coincidência for encontrada o comando default será executado. O default é opcional.

A sequência de comandos é executada até que o comando break seja encontrado. Se não houver o comando break, todos os comandos abaixo serão realizados em sequência, mesmo que estejam declarados em outro bloco de comandos.

# Comandos Condicionais

---

Comparando com algoritmo temos:

escolha variável  
  caso valor1  
    comandos 1  
  caso valor2  
    comandos 2  
  ...  
  outro caso  
    comandos  
fimescolha



```
switch (variável)
{
  case valor1:
    comandos 1;
    break;
  case valor2:
    comandos 2;
    break;
  ...
  default:
    comandos;
}
```

# Comandos Condicionais

---

Ex:

```
#include<stdio.h>
main()
{
    int x;
    printf("1. incluir\n");
    printf("2. alterar\n");
    printf("3. excluir\n");
    printf("Digite sua opção:");
    scanf("%d",&x);
    switch(x)
    {
        case 1:
            printf("escolheu inclusão\n");
            break;
        case 2:
            printf("escolheu alteração\n");
            break;
        case 3:
            printf("escolheu exclusão\n");
            break;
        default:
            printf("opção inválida\n");
    }
}
```



# Comandos de Repetição

---

O comando mais simples de repetição é o **for** que define um valor inicial a uma variável e vai modificando-o automaticamente a cada execução:

```
for(inicialização; condição; incremento)  
    comando;
```

- ✓ Inicialização: atribuição de um valor à variável de controle;
- ✓ Condição: teste que verifica quando o comando de repetição será encerrado;
- ✓ Incremento: modificação que será realizada no valor da variável a cada execução

# Comandos de Repetição

---

Comparando com algoritmo temos:

para *variável* de *valor-inicial* ate *valor-fim* faça  
    comando  
fimpara



for(inicialização; condição; incremento)  
    comando;

# Comandos de Repetição

---

Ex: Imprime valores de 1 a 100 um em cada linha.

```
#include<stdio.h>
main()
{
    int x;
    for(x=1;x<100;x++)
        printf("%d\n",x);
}
```

Obs: para mais de um comando não esquecer de usar a instrução composta com { e }

# Comandos de Repetição

---

Outros exemplos:

- ✓ Para mudar o valor da variável de 2 em 2

```
for(x=1;x<100;x+=2)
    printf("%d\n",x);
```

- ✓ Para usar mais de uma variável de controle no mesmo comando for

```
for (x=0,y=0;x+y<100;++x,++y)
    printf("%d ",x+y);
```

# Comandos de Repetição

---

Outro comando de repetição é o **while** que testa uma condição e executa um comando caso a condição seja verdadeira. Ao chegar no fim do comando um novo teste será executado para se determinar se a repetição continua ou se termina (caso o resultado do teste seja falso):

```
while(condição)
    comando;
```

- ✓ A condição é testada antes de se lançar a execução do comando.

# Comandos de Repetição

---

Comparando com algoritmo temos:

```
enquanto condição faça  
    comando  
fimpara
```



```
while(condição)  
    comando;
```

# Comandos de Repetição

---

Ex: Lê caracteres do teclado até que a letra a seja digitada.

```
#include<stdio.h>
main()
{
    char ch;
    while(ch != 'a')
        scanf("%c",&ch);
}
```

Obs: para mais de um comando não esquecer de usar a instrução composta com { e }

# Comandos de Repetição

---

O último comando de repetição é o **do-while** que executa um comando e depois testa uma condição para se determinar se a repetição continua ou se termina (caso o resultado do teste seja falso):

```
do
{
    comando;
} while(condição);
```

- ✓ Ele realiza sempre pelo menos uma execução do comando já que o teste é só no fim.



# Comandos de Repetição

---

Comparando com algoritmo temos:

```
repita  
    comando  
ate condição
```



```
do  
{  
    comando;  
} while(condição);
```

# Comandos de Repetição

---

Ex: Lê caracteres do teclado até que a letra a seja digitada.

```
#include<stdio.h>
main()
{
    char ch;
    do
    {
        scanf("%c",&ch);
    } while(ch != 'a');
}
```

# Vetor

---

Vetores são listas ordenadas de determinados tipos de dados. Na linguagem C o vetor inicia com índice 0 (primeiro elemento do vetor e vai até o último elemento declarado na variável).

Ex: vetor para armazenar as notas de uma turma.

Notas	6,1	2,3	9,4	5,1	8,9	9,8	10	7,0
Posição	0	1	2	3	4	5	6	7

# Vetor

---

Na linguagem C é declarado da seguinte forma:

**tipo nome\_vetor [tamanho];**

Ex: float nota [8];

char nome\_cliente [50];

Comparando com algoritmo temos:

idade:vetor [1..10] de inteiro



int idade [10];

# Vetor

Ex: Colocar os números de 1 a 5 no vetor.

```
#include<stdio.h>
main()
{
    int vet [5], i;
    for (i=0; i<5; i++)
        vet[i] = i + 1;
}
```

Ex2: Colocar os números pares de 0 a 18 no vetor.

```
#include<stdio.h>
main()
{
    int x [10], t;
    for (t=0; t<10; t++)
    {
        x[t]=t*2;
        printf("%d\n",x[t]);
    }
}
```

# Vetor

---

Ex3: Ler notas de 5 alunos e calcular a média final.

```
#include<stdio.h>
main()
{
    int notas[5],i,soma;
    for(i=0;i<5;i++)
    {
        printf("Digite a nota do aluno %d: ",i);
        scanf("%d",&notas[i]);
    }
    soma=0;
    for(i=0;i<5;i++)
        soma=soma+notas[i];
    printf("Media das notas: %d.",soma/5);
}
```

# String

---

Na linguagem C não existe um tipo de dado string, no seu lugar é utilizado uma matriz de caracteres.

Uma string é um vetor de caracteres, cujo final é indicado com um caractere nulo ('\0').

Sendo assim, ao definir uma string, deve-se levar em consideração, além do número de caracteres da string, o caractere nulo que termina a string.

Ex:

`char nome [11] ➡ "engenharia" + '\0'`

# String

---

Além dos comandos `scanf` e `printf`, no caso das strings existem também os seguintes comandos para:

## Entrada de dados

- **`gets(vetor)`** : lê string até o <enter> e guarda em vetor;

## Saída de dados

- **`puts(mensagem)`** : escreve mensagem na tela e coloca o `\n` no final.



# String

---

Ex: Ler um nome e mostrá-lo em seguida na tela.

```
#include<stdio.h>
main()
{
    int nome[50];
    printf("Digite um nome: ");
    gets(nome);
    printf("Nome: %s.",nome);
}
```

# String

---

A biblioteca **<string.h>** inclui vários comandos de manipulação de strings como:

- **strcpy(string1,string2)** : copia a string1 em string2;
- **strcat(string1,string2)** : concatena as duas strings em string1;
- **strcmp(string1,string2)** : compara as duas strings.