

Aulas Unidade 1

Algoritmos e Programação

Universidade Federal do Rio Grande do Norte – UFRN

Escola Agrícola de Jundiaí – EAJ

Tecnologia em Análise e Desenvolvimento de Sistemas – TADS

Período Letivo 2024.1

Profa. Alessandra Mendes Pacheco (alemendes@gmail.com)

Metodologia

- Disciplina 100% prática.
- Aulas expositivas
 - Material audiovisual;
 - Discussões e análises;
 - Resoluções de questões.
- Avaliações escritas e/ou orais.
 - Listas de exercícios;
 - Resoluções orais;
 - Aplicação de provas escritas.
- Dúvidas:
 - Utilização de e-mail (alemendes@gmail.com) e WhatsApp;
 - Monitoria a definir (?).

Referências

- DEITEL, H.M. C++ : como programar. São Paulo: Pearson Prentice Hall, 2006.
- PEREIRA, S. L. Linguagem C++. São Paulo: FATEC, 1999.
- DAURICIO, J. S. Algoritmos e lógica de programação. Londrina : Editora e Distribuidora Educacional S.A., 2015.
- FORBELLONE, A. L. V. EBERSPACHER H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3. ed. São Paulo : Prentice Hall, 2005.
- Apostilas de domínio público.
- Slides utilizados no curso.
- Códigos-fonte dos algoritmos desenvolvidos.

Ementa do Curso

1. **Introdução a algoritmos: definições, características e formas de representação;**
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

Ementa do Curso

- 1. Introdução a algoritmos: definições, características e formas de representação;**
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

Definições – um pouco de história

- ▶ A palavra “**algoritmo**” surgiu na Idade Média e vem do nome do persa Muḥammad ibn Musa **al-Khwarizmi**, que foi astrônomo na Casa de Sabedoria do Califado Abássida, em Bagdá. Graças a sua vasta obra, o sistema de numeração indo-arábico, que usamos até hoje, se difundiu no Oriente Médio e no Ocidente.
- ▶ O nome “al-Khwarizmi”, devidamente latinizado, primeiro foi associado ao **sistema de numeração** e depois ao conceito moderno de algoritmo.
- ▶ A ideia principal contida na palavra refere-se à descrição sistemática da maneira de se realizar alguma tarefa.
- ▶ Para a Ciência da computação, o conceito de algoritmo foi formalizado em 1936 por Alan Turing (Máquina de Turing) e Alonzo Church, que formaram as primeiras fundações da Ciência da computação, sendo esta formalização descrita a seguir:

Um algoritmo é um conjunto não ambíguo e ordenado de passos executáveis que definem um processo finito.

Definições

- Um algoritmo é simplesmente uma "**receita**" para a execução de uma tarefa ou resolução de um problema.
- Como toda receita, um algoritmo também deve ser **finito**.
- Um algoritmo é uma **sequência de raciocínios**, instruções ou operações para alcançar um **objetivo**, ou seja, representa uma solução para um problema.

Compute-it

Entendendo um algoritmo...

<http://compute-it.toxicode.fr/>



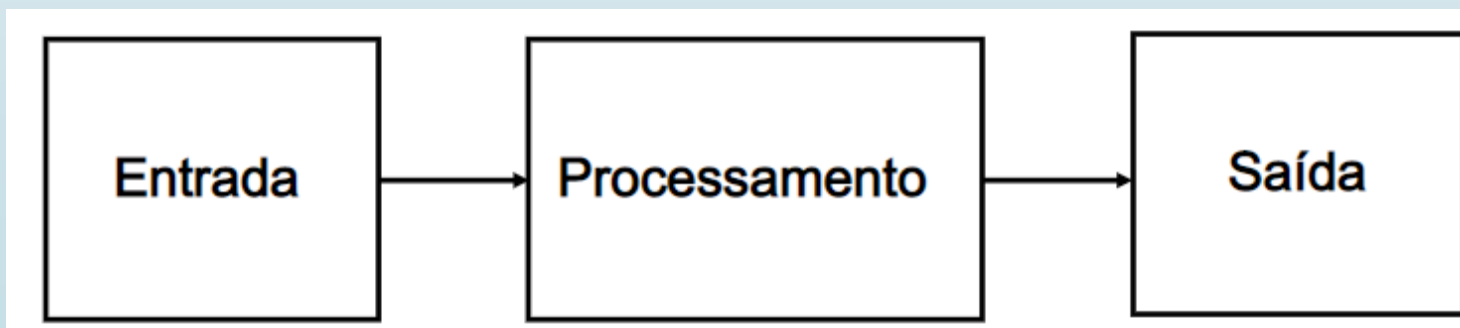
Definições

Um algoritmo é um conjunto das regras e procedimentos lógicos perfeitamente definidos que levam à solução de um problema em um número finito de etapas.

- Um algoritmo é simplesmente uma "**receita**" para a execução de uma tarefa ou resolução de um problema.
 - Como toda receita, um algoritmo também deve ser **finito**.
- Um algoritmo é uma **sequência de raciocínios, instruções ou operações** para alcançar um objetivo, ou seja, representa uma **solução** para um problema.

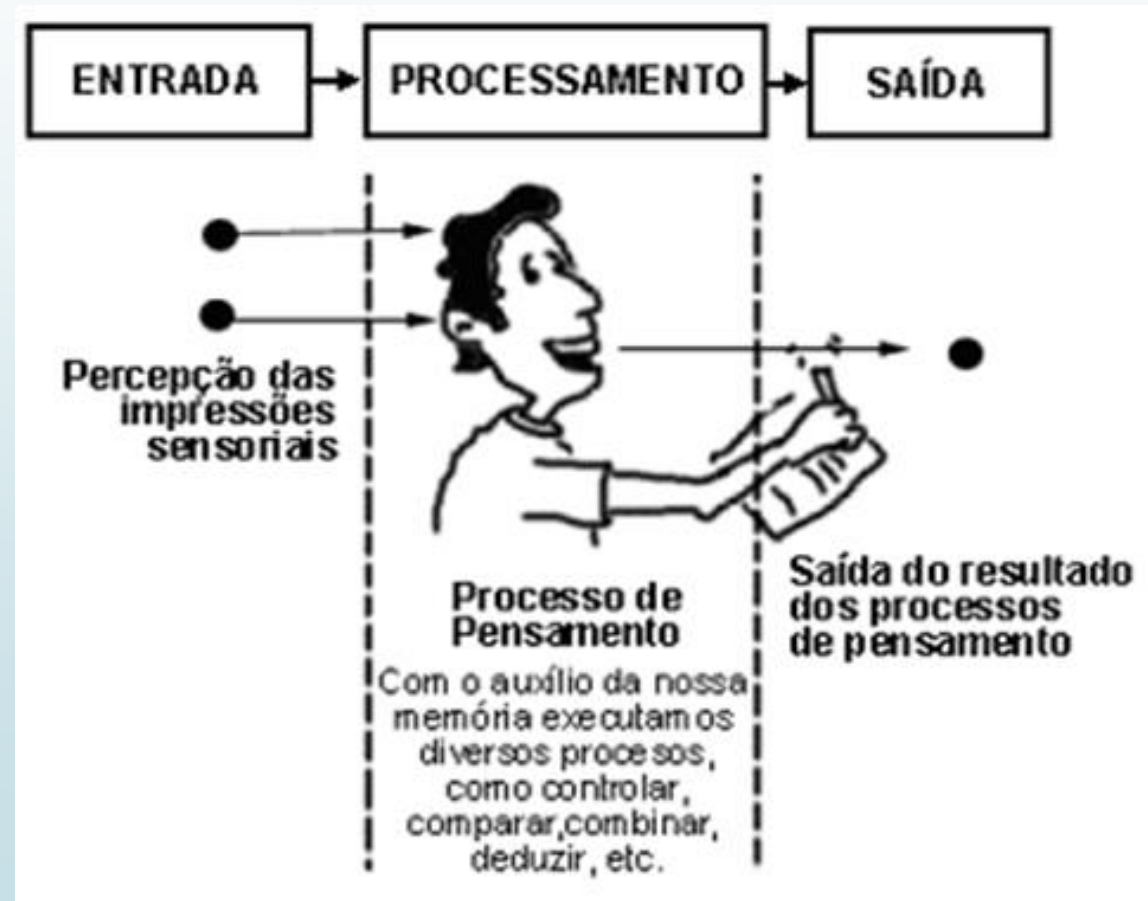
Características

- Um algoritmo conta com a **entrada** (*input*) e **saída** (*output*) de informações **mediadas pelas instruções**. Assim, todo problema a ser codificado deve ser **dividido em 3 fases**:
 - **Entrada**: são os dados e informações iniciais do problema
 - **Processamento**: são os procedimentos utilizados para solucionar o problema
 - **Saída**: são os dados e informações resultantes do processamento.



Características

- Fazendo uma analogia...



Características

► Onde estão as fases?

ENTRADA

Ingredientes da receita, dados e informações necessários para o resultado final.

PROCESSAMENTO

Modo de fazer, passo-a-passo, conjunto de instruções necessárias para que o objetivo seja atingido.

SAÍDA

O produto final, o bolo.

BOLO DE FUBÁ

INGREDIENTES:

- 2 XÍCARAS DE AÇÚCAR
- 3 GEMAS
- 3 CLARAS BATIDAS EM NEVE
- 1 XÍCARA DE ÓLEO
- 2 XÍCARAS DE FARINHA DE TRIGO
- 1 XÍCARA DE FUBÁ
- 1 COPO DE LEITE
- 1 COLHER DE FERMENTO

MODO DE FAZER:

COLOQUE O AÇÚCAR E A GEMA NA BATEDEIRA E BATA ATÉ FICAR CLARO, EM SEGUIDA ACRESCENTE O ÓLEO E DESLIGUE A BATEDEIRA. VÁ COLOCANDO AOS POUCOS A FARINHA E O FUBÁ, MEXENDO COM UMA COLHER. ACRESCENTE O LEITE, AS CLARAS EM NEVE E, POR ÚLTIMO, O FERMENTO. DESPEJE NUMA ASSADEIRA UNTADA E ASSE EM FORNO PRÉ-AQUECIDO.



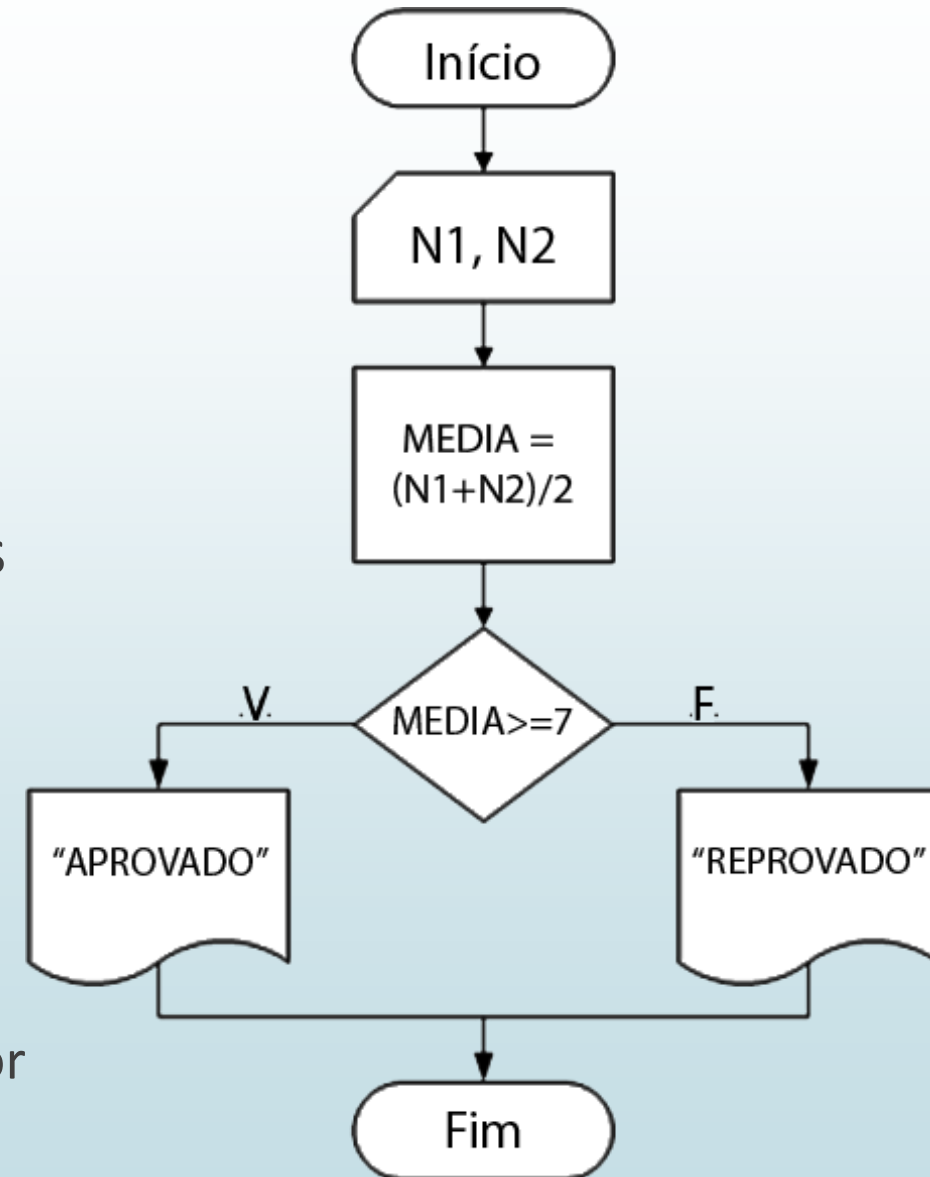
Formas de representação

- Um algoritmo pode ser representado como **descrição narrativa, fluxograma ou pseudocódigo**.
- **Descrição narrativa**: utiliza a linguagem natural para especificar os passos para a realização das tarefas.
- Exemplo: Trocar uma lâmpada
 - 1: Pegar uma escada
 - 2: Posicionar a escada embaixo da lâmpada
 - 3: Buscar uma lâmpada nova
 - 4: Subir na escada com a lâmpada nova
 - 5: Retirar a lâmpada velha
 - 6: Colocar a lâmpada nova



Formas de representação

- **Fluxogramas:** utiliza-se figuras geométricas para ilustrar os passos a serem seguidos na resolução de problemas.
- Exemplo: Cálculo da média de duas notas
 - Entrada: duas notas (N1 e N2)
 - Processamento: calcular a média aritmética $(N1+N2)/2$ e verificar se a média é maior ou igual a 7.
 - Saída: “Aprovado” (se a condição for verdadeira) ou “Reprovado” (se a condição for falsa).



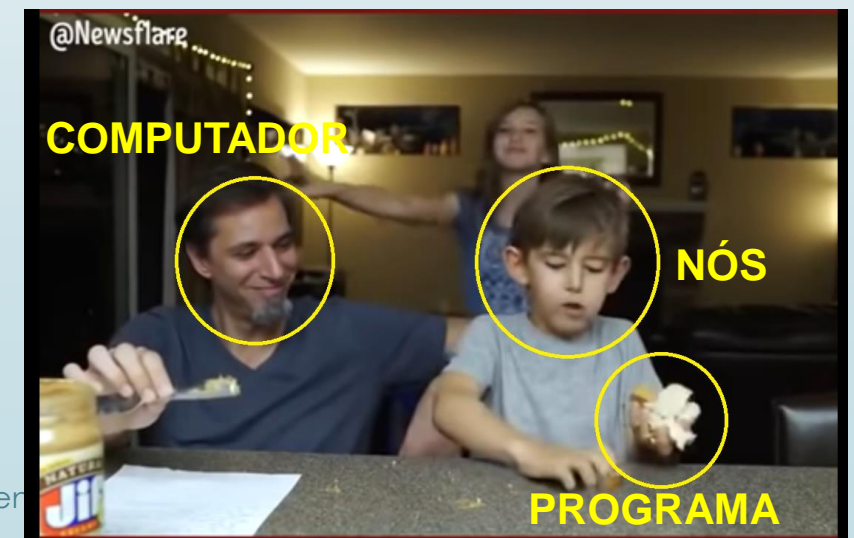
Formas de representação

- **Pseudocódigo:** forma de representação de algoritmos rica em detalhes. Utiliza linguagem estruturada e se assemelha, na forma, a um programa escrito na linguagem de programação.
- É um tipo de algoritmo que utiliza uma linguagem flexível intermediária entre a linguagem natural e a linguagem de programação.
 - Pseudocódigo significa “falso código”.

```
Algoritmo Media;  
    Var N1, N2, MEDIA: real;  
Início  
    Leia (N1, N2);  
    MEDIA ← (N1 + N2) / 2;  
    Se MEDIA >= 7 então  
        Escreva "Aprovado"  
    Senão  
        Escreva "Reprovado";  
    Fim_se  
Fim
```


Formas de representação

- Do pseudocódigo (algoritmo) ao programa... E agora?
- Quando transformamos uma sequência de instruções em um algoritmo utilizando uma linguagem de programação, geramos um **código-fonte**.
- Este código-fonte (arquivo texto) precisa ser **compilado** para que se torne um **executável** (linguagem de máquina).
- O computador “rodará” o executável **exatamente** como foi programado e na sequência exata.
- *Se não funcionou como deveria...*



COMO FAZER UM SANDUÍCHE

@Newsflare



DESAFIO DE INSTRUÇÕES EXATAS

INCRÍVEL

Ementa do Curso

1. Introdução a algoritmos: definições, características e formas de representação;
- 2. Estrutura geral de um programa**
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros**
 - 2.2. Escopo**
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

A linguagem de programação C++

- O C++ é uma linguagem de programação de nível médio, **baseada na linguagem C**, e foi criada para ser tão eficiente quanto o C, porém com **novas funções**.
 - Seu desenvolvimento começou na década de 80, por Bjarne Stroustrup.
 - A primeira versão oficial do C++ apareceu em 1985, juntamente com o livro “*The C++ Programming Language*”.
 - Em 1989 foi lançada uma segunda versão da linguagem, com acréscimo de características de Orientação a Objetos.
- O C++ é uma linguagem que suporta **múltiplos paradigmas**.
- Muitos códigos podem ser **transferidos** para C facilmente, pois o C++ foi criado para ter **compatibilidade** com o C.
- **Não é necessário** um ambiente de desenvolvimento muito potente para o desenvolvimento de C++.

Estrutura básica de um programa em C++

```
exemplo.cpp •
exemplo.cpp > ...
1  // isso é um comentário e não será compilado
2
3  /* isso é um comentário
4  de múltiplas linhas
5  e também não será compilado
6  */
7
8  /*
9  O cabeçalho #include<> indica ao compilador todas as bibliotecas que serão incluídas e que
10 o programa utilizará. O #include <iostream> inclui a biblioteca iostream que contém as
11 principais funções, comandos e classes de entrada e saída do C++ necessárias para programas
12 que, por exemplo, recebam dados via teclado e enviem dados via monitor (cin, cout).
13 */
14 #include<iostream>
15
16 using namespace std; // é uma directiva de utilização do espaço std
17 //(serve por ex para se escrever simplesmente cout em vez de std::cout)
18
19 //Em todo programa C++, deve existir uma única função chamada main
20 int main(){ // A função main marca o ponto de partida do algoritmo e as {} a delimitam
21     // declaração de variáveis
22     // instruções em C++
23 }
```

○ compilador online Repl.it

The screenshot displays the Repl.it online C++ compiler interface. The browser address bar shows `repl.it/languages/cpp`. The page header includes the Repl.it logo, the text "C++ Online Compiler, IDE, Editor, Interpreter and REPL", and a description: "Code, collaborate, compile, run, share, and deploy C++ online from your browser". Navigation buttons for "Save" and "Run" are visible. A "Sign up" button is in the top right corner.

The left sidebar shows a "Files" panel with a single file named `main.cpp`. The main editor area displays the following C++ code:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main() {
6      cout << "Hello World!\n";
7  }
```

The right panel shows the terminal output of the compilation and execution:

```
clang++-7 -pthread -std=c++17 -o main main.cpp
./main
Hello World!
```

Escopo

- Quando se declara um elemento em um algoritmo, seu nome **só pode ser "visto" e usado** em determinadas **partes** do seu programa.
- O contexto no qual um nome ou elemento é visível é chamado de **escopo**.
- Assim, escopos são contextos delimitantes aos quais os elementos estão associados.
- Em outras palavras, o escopo é utilizado para definir o **grau de ocultação da informação**, isto é, a visibilidade e acessibilidade aos elementos em diferentes partes do programa.

Escopo

- Variáveis e funções, por exemplo, podem estar localizadas em dois escopos: **global** e **local**.
 - No escopo global o elemento é acessível em qualquer local a partir do ponto em que foi criado.
 - No escopo local ele só é acessível no local onde foi declarado e a partir do ponto em que foi criado.



- Por causa disso pode-se ter dois elementos com o mesmo nome em escopos diferentes.

Escopo

➡ Quais as visibilidades?



Programa Principal:

A,B

Rotina 1:

A,X

Rotina 1.1:

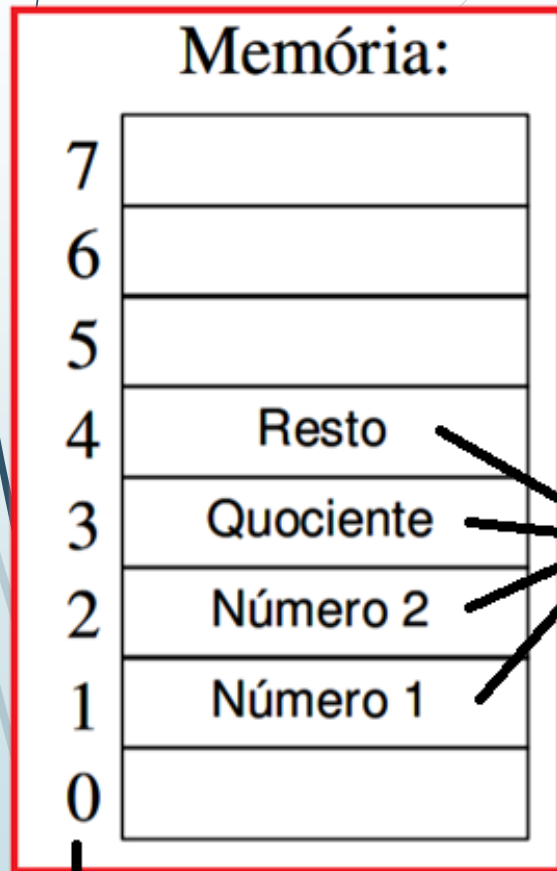
Y

Rotina 2:

X

Ementa do Curso

1. Aspectos históricos, conceitos e características básicas;
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
- 3. Variáveis simples e tipos primitivos**
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

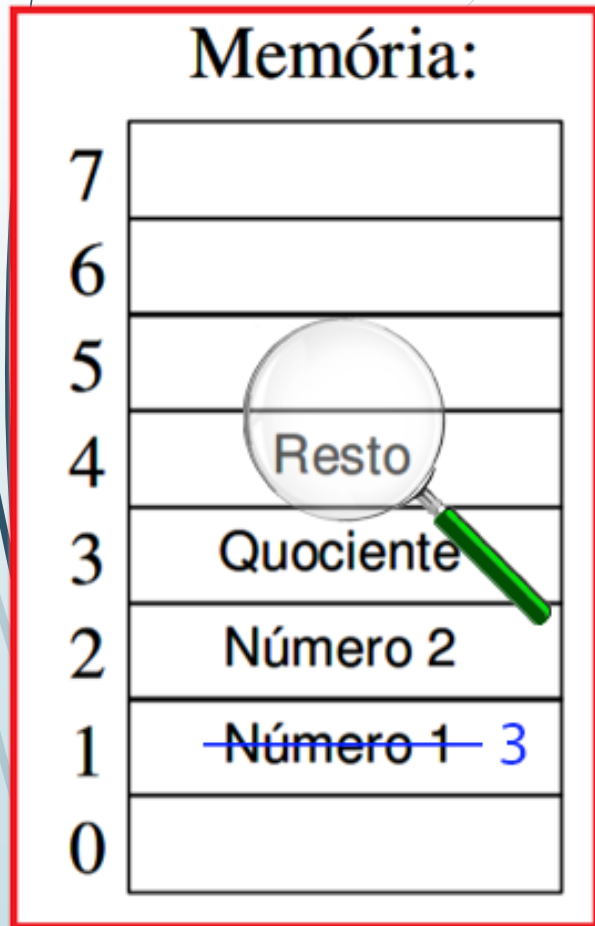


CONTEÚDO

ENDEREÇO

- A memória de um computador é o local em que são **guardadas** as informações necessárias para execução de programas.
- A memória pode ser entendida como uma **sequência de células** nas quais se podem armazenar dados (**conteúdo**).
- Como as células estão ordenadas, atribui-se a cada uma delas um número (**endereço**) correspondente a sua posição na sequência.

Assim, cada célula de memória é caracterizada pelo seu endereço e seu conteúdo.



LEITURA

ESCRITA

- Esta memória é **volátil** (uma vez desligado o computador, as informações são perdidas).
- Os algoritmos utilizam esta memória para **salvar informações durante a sua execução**.
- O computador **manipula as informações** contidas em sua memória através da leitura e da escrita.
 - A **leitura** localiza a célula correspondente ao endereço desejado e consulta o valor armazenado (o valor não é alterado).
 - A **escrita** localiza a célula correspondente ao endereço desejado e substitui o seu conteúdo pelo novo valor (o conteúdo anterior é perdido de forma irreversível).

Variáveis



~ MEMÓRIA Conjunto de locais para armazenar os dados.
Os dados podem ser armazenados na memória por meio das variáveis.

~ VARIÁVEL Local onde um dado específico é guardado.
Através do nome da variável, o seu dado pode ser acessado (leitura) e modificado (escrita) pelo algoritmo.

~ DADO Valor de um tipo específico que é armazenado em uma variável.

Variáveis

- Uma variável possui **NOME**, **TIPO** e **CONTEÚDO**.
 - *As regras para nomes de variáveis mudam de uma linguagem para outra. Normalmente não podem começar por números, não podem ter letras que não pertençam ao alfabeto inglês, não podem ter espaços, não podem ser palavras reservadas da linguagem de programação e não podem possuir caracteres especiais (exceto “_”).*
- Variáveis devem ser **declaradas antes** de serem utilizadas e ao declarar uma variável, o computador **reserva um espaço na memória** para ela.
 - Cada tipo de variável ocupa um **tamanho diferente** na memória.
- Exemplo de declaração de variável:

```
Tipo nome_da_variável;
```

Variáveis

- O nome da variável é **único** em todo o algoritmo.
 - A variável “NOME” é diferente da variável “nome”.
- O **conteúdo** da variável deve ser **do mesmo tipo** usado na **declaração** da variável.
- O **uso** de uma variável (leitura) em uma expressão representa o seu conteúdo naquele momento.
- Na atribuição, o conteúdo da variável é **substituído** por outro (escrita).

*Uma constante é uma “variável especial” pois também reservará um espaço de memória para o seu dado. Porém, uma constante armazenará um valor **ÚNICO**, que **NÃO** mudará durante o algoritmo.*

Tipos primitivos

- Definem o tipo de dado que poderá ser armazenado naquela variável.

Uma vez declarado o tipo, este não poderá ser modificado.

- **Inteiro** (`int`): números positivos e negativos sem casas decimais
- **Real** (`float`): números positivos e negativos que possuem casas decimais
 - *Obs: o tipo `double` é um `float` com maior precisão nas casas decimais*
- **Caracter** (`char`): caracteres simples
 - As sequencias de caracteres são *strings* (a detalhar...).
- **Lógico** (`bool`): verdadeiros (*true*) ou Falsos (*false*).

- Exemplo de declaração de variável em C++:

```
int idade;
```

Ementa do Curso

1. Aspectos históricos, conceitos e características básicas;
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
- 4. Operadores**
 - 4.1. Atribuição**
 - 4.2. Operadores aritméticos**
 - 4.3. Operadores lógicos**
 - 4.4. Operadores relacionais**
5. Comandos de entrada e saída
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

Operadores – Atribuição

► **Atribuição (=):** Permite atribuir uma expressão à uma variável.

► **Sintaxe:** <nome_da_variável> = <expressão>

► A expressão à direita da “=”
é calculada;

► O valor resultante é inserido
na variável à esquerda da “=”.

► É possível atribuir o mesmo valor a múltiplas variáveis em uma única instrução. Exemplo:

```
int x, y, z;  
x = y = z = 0;
```

VÁLIDO	INVÁLIDO
a = 1; delta = b * b - 4*a*c; i = j;	1 = a; a+b = 10;

Operadores aritméticos

- Precedência (ordem em que os operadores são avaliados):
 - São avaliados **da esquerda para a direita na expressão**, obedecendo as prioridades de avaliação. Parênteses podem ser utilizados para quebrar a precedência.
- Prioridades de avaliação (do maior para o menor):

++, --

*, /, %

+, -

- Exemplos:

`a = 2 + 3 * 5;` Resultado: **a = 17**

`a = (2 + 3) * 5;` Resultado: **a = 25**

OPERADOR	OPERAÇÃO
+	ADIÇÃO
-	SUBTRAÇÃO
*	MULTIPLICAÇÃO
/	DIVISÃO
%	RESTO DA DIVISÃO INTEIRA
++	INCREMENTO
--	DECREMENTO

Operadores aritméticos

■ Restrições:

- Os operadores +, -, *, / podem operar sobre **números inteiros ou reais**;
- O operador % aceita apenas operandos **inteiros**;
- O **denominador** em uma operação de divisão deve ser **diferente de 0**.

■ Exemplos:

6 + 2

Resultado: 8

2.0 + 2.0

Resultado: 4.0

6.5 % 2

inválido, 6.5 não é int

10 / 0

inválido, divisão por 0

Operadores aritméticos

- **Operadores unários de incremento (++) e decremento (--):**
 - ++ soma 1 ao seu operando (incremento)
 - -- subtrai 1 do seu operando (decremento)
- Podem ser utilizados como **prefixo** (incremento/decremento anterior) ou **sufixo** (incremento/decremento posterior) do operando em uma expressão.
 - x++ : usa o valor de x e **posteriormente** o incrementa
 - ++x : incrementa o valor de x **antes** do uso do seu valor
- Exemplos:

`x=10; y = ++x;` Resultado: `x = 11` e `y == 11`

`x=10; y = x++;` Resultado: `x = 11` e `y == 10`

Operadores aritméticos

► Operadores de atribuição aritmética:

OPERADOR	EQUIVALÊNCIA
var += exp	var = var + exp
var -= exp	var = var - exp
var *= exp	var = var * exp
var /= exp	var = var / exp

► Exemplos:

num *= 3

EQUIVALE A

num = num * 3

quantia /= 10

EQUIVALE A

quantia = quantia / 10

Operadores relacionais e lógicos

► Operadores Relacionais:

OPERADOR	OPERAÇÃO
>	MAIOR
<	MENOR
>=	MAIOR OU IGUAL
<=	MENOR OU IGUAL
==	IGUAL
!=	DIFERENTE

► Operadores Lógicos:

OPERADOR	OPERAÇÃO
&&	“E” Lógico (AND)
	“OU” Lógico (OR)
!	“Não” Lógico (NOT)

Operadores relacionais e lógicos

- O tipo *bool* (em C++) pode receber os valores **true** (verdadeiro) ou **false** (falso)
- Porém, o resultado de uma expressão lógica ou relacional é um valor numérico
 - O “verdadeiro” é representado pelo valor 1 (um).
 - O “falso” é representado pelo valor 0 (zero).
- Portanto...

Variável lógica a	Variável lógica b	a&&b	a b	!a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	0	0	0	1

Operadores relacionais e lógicos

- Ordem de precedência (da maior para a menor):

!

>, <, >=, <=,

==, !=

&&

||

- Exemplos:

```
bool x;
```

```
x = 1 || 0 && 0;
```

```
x = ?
```



Ementa do Curso

1. Aspectos históricos, conceitos e características básicas;
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
- 5. Comandos de entrada e saída**
6. Estruturas de controle de fluxo de execução de programas
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição

Comando de saída

- A operação de saída permite que o **programa forneça dados ao usuário**.

- Sintaxes de saída (em C++):

```
cout << "oi!";  
cout << "a = " << a << "e fim";  
cout << "\nteste1\n\tteste2";  
cout << "nova linha" << endl << "ok";
```

- Exemplo (em C++):

```
#include<iostream>  
using namespace std;  
int main() {  
    cout << "Oi mundo!";  
    system("pause");  
}
```

Comando de entrada

- A operação de entrada permite que o **usuário forneça dados ao programa**.

- Sintaxes de entrada (em C++):

```
cin >> a;
```

```
cin >> a >> b >> c;
```

- Exemplo (em C++):

```
#include<iostream>
using namespace std;
int main() {
    int idade;
    cout << "Digite a sua idade: ";
    cin >> idade;
    system("pause");
}
```

Ementa do Curso

1. Aspectos históricos, conceitos e características básicas;
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
- 6. Estruturas de controle de fluxo de execução de programas**
 - 6.1. Comandos de seleção**
 - 6.2. Comandos de repetição

Estrutura condicional simples

■ Em português:

se (condição) **então**

<comando>

fim-se

■ Em C++:

if (condição)

<comando>;

A **condição** deve ser
uma expressão
lógica!

O **comando** só será
executado se a
condição for
verdadeira!

Estrutura condicional simples

► Em português:

se (condição) **então**

<comando>

fim-se

► Em C++:

if (condição)

<comando>;

Se **mais de um comando** deve ser executado quando a condição for verdadeira, esses comandos devem ser transformados em um **comando composto**.

Estrutura condicional simples

- Comando composto: consiste em vários comandos que devem ser executados em uma ordem específica.
- Em C, o comando composto deve ficar reunido entre chaves { e }.

```
if (condição) {  
    <comando1>;  
    <comando2>;  
}
```

Comando Composto

Estrutura condicional simples

► Exemplo (em C++):

Chaves
opcionais, pois
só tem um
comando
interno ao `if`
(não é
composto).

```
#include<iostream>
using namespace std;
int main(){
    int idade;
    cout << "Digite a sua idade: ";
    cin >> idade;
    if(idade >= 18) {
        cout << "Pode dirigir";
    }
    system("pause");
}
```

Estrutura condicional composta

► Em português:

se (condição) **então**

<comando1>

senão

<comando2>

fim-se

A **condição** deve ser uma **expressão lógica**.

* Se a condição for **verdadeira**, será executado o **comando1** e não será executado o comando2.

* Se a condição for **falsa**, será executado o **comando2** e não será executado o comando1.

Estrutura condicional composta

► Em C++:

```
if (condição) {  
    <comando1>;  
    <comando2>;  
} else {  
    <comando3>;  
    <comando4>;  
}
```

Se **mais de um comando** deve ser executado quando a condição for verdadeira ou quando a condição for falsa, esse conjunto de comandos deve ser transformado em um **comando composto**.

Estrutura condicional composta

► Exemplo (em C++):



```
#include<iostream>
using namespace std;
int main(){
    int idade;
    cout << "Digite a sua idade: ";
    cin >> idade;
    if(idade >= 18) {
        cout << "Pode dirigir";
    }else
        cout << "Ainda não pode dirigir";
    system("pause");
}
```

Estrutura de seleção múltipla

► Em português:

escolha (variável)

caso <valor1> **faça**

<comando1>

caso <valor2> **faça**

<comando2>

caso <valor3> **faça**

<comando3>

fim_escolha

O **conteúdo** de uma variável é **comparado** com um **valor** constante, e caso a comparação seja **verdadeira**, um determinado comando é **executado**.

Estrutura de seleção múltipla

► Em C++:

```
switch (variável) {  
    case <valor1>: <comando1>;  
        break;  
    case <valor2>: <comando2>;  
        break;  
    default:  
        <comando3>;  
}
```

- **break**: termina a execução do *switch* e o programa continua a executar na instrução seguinte, evitando testar as demais alternativas de forma desnecessária quando uma opção verdadeira já foi encontrada.

- **default**: seus comandos serão executados quando nenhuma das alternativas anteriores for verdadeira.

if x switch

```
// exemplo do uso do if a partir do main()
int main(){
    int valor;
    cout << "Digite um valor de 1 a 7: ";
    cin >> valor;
    if(valor==1)
        cout << "Domingo\n";
    else
        if(valor==2)
            cout << "Segunda\n";
        else
            if(valor==3)
                cout << "Terça\n";
            else
                if(valor==4)
                    cout << "Quarta\n";
                else
                    if(valor==5)
                        cout << "Quinta\n";
                    else
                        if(valor==6)
                            cout << "Sexta\n";
                        else
                            if(valor==7)
                                cout << "Sabado\n";
                            else
                                cout << "Valor invalido!\n";
    system("pause");
}
```

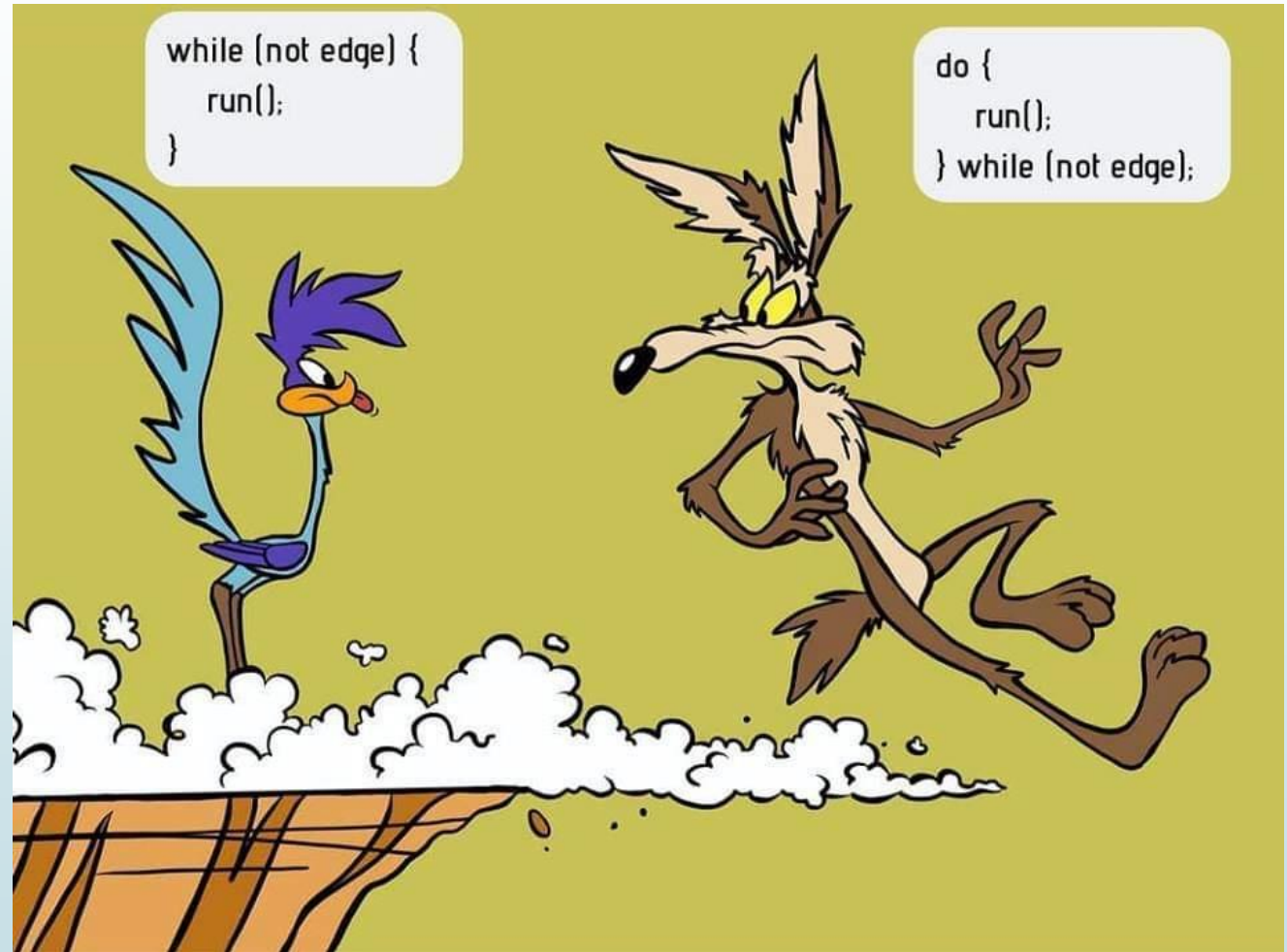
```
// exemplo do uso do switch a partir do main()
int main(){
    int valor;
    cout << "Digite um valor de 1 a 7: ";
    cin >> valor;
    switch(valor) {
        case 1: cout << "Domingo\n";
                break;
        case 2: cout << "Segunda\n";
                break;
        case 3: cout << "Terça\n";
                break;
        case 4: cout << "Quarta\n";
                break;
        case 5: cout << "Quinta\n";
                break;
        case 6: cout << "Sexta\n";
                break;
        case 7: cout << "Sabado\n";
                break;
        default: cout << "Valor invalido!\n";
    }
    system("pause");
}
```

Ementa do Curso

1. Aspectos históricos, conceitos e características básicas;
2. Estrutura geral de um programa
 - 2.1. Características da linguagem adotada, do ambiente de programação e rastreamento de erros
 - 2.2. Escopo
3. Variáveis simples e tipos primitivos
4. Operadores
 - 4.1. Atribuição
 - 4.2. Operadores aritméticos
 - 4.3. Operadores lógicos
 - 4.4. Operadores relacionais
5. Comandos de entrada e saída
- 6. Estruturas de controle de fluxo de execução de programas**
 - 6.1. Comandos de seleção
 - 6.2. Comandos de repetição**

Estruturas de Repetição – Laços

São estruturas que permitem a execução mais de uma vez, de um mesmo comando ou conjunto de comandos, de acordo com uma condição.



Estruturas de Repetição – Laços

- Repetição com teste no início (*while*);
- Repetição com teste no fim (*do_while*);
- Repetição contada (*for*).

Permitem que uma sequência de comandos seja executada **repetidamente** até que uma determinada **condição de interrupção** seja satisfeita.

Repetição com teste no início

■ Em português:

enquanto (condição) **faça**

<comando1>

<comando2>

<comando3>

fim-enquanto

Condição para **continuar** a repetição, ou seja, sempre que a condição for **verdadeira**, a repetição continuará.

Repetição com teste no início

► Em C++:

```
while (condição) {  
    <comando1>;  
    <comando2>;  
    <comando3>;  
}
```

Exemplo 1 - While

- Elabore um algoritmo que leia a idade de uma pessoa. Enquanto o valor fornecido não estiver correto, (maior ou igual a 1) escreva uma mensagem de erro e solicite novamente a idade. Quando o valor fornecido estiver correto, calcule e escreva quantos anos faltam para chegar aos 40 anos, se for o caso.

Exemplo 1 - While

```
int main() {
    int idade;
    cout << "\nDigite a idade: ";
    cin >> idade;
    while(idade<=0) {
        cout << "\nIdade invalida! ";
        cout << "\nDigite a idade: ";
        cin >> idade;
    }
    if(idade<40)
        cout << "\nFaltam "<< (40-idade) << " anos para 40 anos";
    else
        cout << "\nJa completou 40 anos";
}
```

A variável de controle (idade) deve ter um valor conhecido (neste caso ela foi lida antes).

A variável de controle deve ter seu valor modificado dentro do laço (neste caso ela foi lida novamente).

Exemplo 2 - While

- Elabore um algoritmo que leia N números reais (o valor de N também deve ser lido), calcule e escreva a soma desses números.

Exemplo 2 - While

```
int main(){
    int N, cont;
    float num, soma = 0;
    cout << "\nQual o total de numeros que serao lidos? ";
    cin >> N;
    cont = 1;
    while(cont<=N){
        cout << "\nDigite um numero: ";
        cin >> num;
        soma = soma + num;
        cont++;
    }
    cout << "\nSoma dos numeros: " << soma;
}
```

Exemplo 3 - While

- Elabore um algoritmo que leia um conjunto de valores correspondentes aos pontos que os alunos obtiveram em um teste. A leitura deverá parar quando o valor fornecido for um número negativo. Ao final, calcule e escreva quantos alunos fizeram o teste, quantos tiveram pontuação alta (pontos ≥ 7) e quantos tiveram pontuação baixa (pontos < 7).

Exemplo 3 - While

```
int main() {
    int pontos, totalAlunos=0, totAlta=0, totBaixa=0;
    cout << "\nDigite os pontos: ";
    cin >> pontos;
    while(pontos>=0) {
        totalAlunos++;
        if(pontos>=7)
            totAlta++;
        else
            totBaixa++;
        cout << "\nDigite os pontos: ";
        cin >> pontos;
    }
    cout << "\nTotal de alunos: " << totalAlunos;
    cout << "\nPontuacao alta: " << totAlta;
    cout << "\nPontuacao baixa: " << totBaixa;
}
```


Repetição com teste no final

■ Em português:

repita

<comando1>

<comando2>

<comando3>

enquanto (condição)

Os comandos são executados **pelo menos uma vez**.

Repetição com teste no final

► Em C++:

do {

<comando1>;

<comando2>;

<comando3>;

} while (condição);

A repetição
acontecerá
enquanto a
condição for
verdadeira.

Exemplo 1 - do_while

- Elabore um algoritmo que leia a idade de uma pessoa. Enquanto o valor fornecido não estiver correto, (maior ou igual a 1) escreva uma mensagem de erro e solicite novamente a idade. Quando o valor fornecido estiver correto, calcule e escreva quantos anos faltam para chegar aos 40 anos, se for o caso.

Exemplo 1 - do_while

```
int main() {
    int idade;
    do{
        cout << "\nDigite a idade: ";
        cin >> idade;
        if(idade<=0)
            cout << "\nIdade invalida";
    } while(idade<=0);
    if(idade<40)
        cout << "\nFaltam " << (40-idade) << " anos para 40 anos";
    else
        cout << "\nJa completou 40 anos";
}
```

Exemplo 2 - do_while

- ▶ Elabore um algoritmo que leia N números reais (o valor de N também deve ser lido), calcule e escreva a soma desses números.

Exemplo 2 - do_while

```
int main() {
    int N, cont, num, soma = 0;
    cout << "\nQual o total de numeros que serao lidos? ";
    cin >> N;
    cont = 0;
    do{
        if(N>0) {
            cout << "\nDigite um numero: ";
            cin >> num;
            soma = soma + num;
            cont++;
        }
    } while(cont<N);
    cout << "\nSoma dos numeros: " << soma;
}
```

Exemplo 3 - do_while

- Elabore um algoritmo que leia um conjunto de valores correspondentes aos pontos que os alunos obtiveram em um teste. A leitura deverá parar quando o valor fornecido for um número negativo. Ao final, calcule e escreva quantos alunos fizeram o teste, quantos tiveram pontuação alta (pontos ≥ 7) e quantos tiveram pontuação baixa (pontos < 7).

Exemplo 3 - do_while

```
int main() {
    int pontos, totalAlunos=0, totAlta=0, totBaixa=0;
    do{
        cout << "\nDigite os pontos: ";
        cin >> pontos;
        if(pontos>=0){
            totalAlunos++;
            if(pontos>=7)
                totAlta++;
            else
                totBaixa++;
        }
    }while(pontos>=0);
    cout << "\nTotal de alunos: " << totalAlunos;
    cout << "\nPontuacao alta: " << totAlta;
    cout << "\nPontuacao baixa: " << totBaixa;
}
```


Repetição contada

► Em português:

para (variável) de (início) até (fim) [PASSO (valor)] faça

<comando1>

<comando2>

<comando3>

fim-para

Comandos a serem executados repetidamente até a (variável) atingir o valor final (fim).

Repetição contada

► Em C++:

```
for (variável=início; condição de parada; passo){  
    <comando1>;  
    <comando2>;  
    <comando3>;  
}
```

O passo pode ser um
incremento ou um
decremento.

Exemplo 1 - for

- Elabore um algoritmo que leia a idade de uma pessoa. Enquanto o valor fornecido não estiver correto, (maior ou igual a 1) escreva uma mensagem de erro e solicite novamente a idade. Quando o valor fornecido estiver correto, calcule e escreva quantos anos faltam para chegar aos 40 anos, se for o caso.

NÃO É POSSÍVEL POIS NÃO CONHECEMOS A CONTAGEM!

Exemplo 2 - for

- Elabore um algoritmo que leia N números reais (o valor de N também deve ser lido), calcule e escreva a soma desses números.

É POSSÍVEL POIS CONHECEMOS A CONTAGEM!

Exemplo 2 - for

```
int main() {
    int N, cont, num, soma = 0;
    cout << "\nQual o total de numeros que serao lidos? ";
    cin >> N;
    for(cont=1; cont<=N; cont++){
        cout << "\nDigite um numero: ";
        cin >> num;
        soma = soma + num;
    }
    cout << "\nSoma dos numeros: " << soma;
}
```

Exemplo 3 - for

- Elabore um algoritmo que leia um conjunto de valores correspondentes aos pontos que os alunos obtiveram em um teste. A leitura deverá parar quando o valor fornecido for um número negativo. Ao final, calcule e escreva quantos alunos fizeram o teste, quantos tiveram pontuação alta (pontos ≥ 7) e quantos tiveram pontuação baixa (pontos < 7).

NÃO É POSSÍVEL POIS NÃO CONHECEMOS A CONTAGEM!

Vamos à prática...

Dúvidas?