

Introdução à Algoritmos

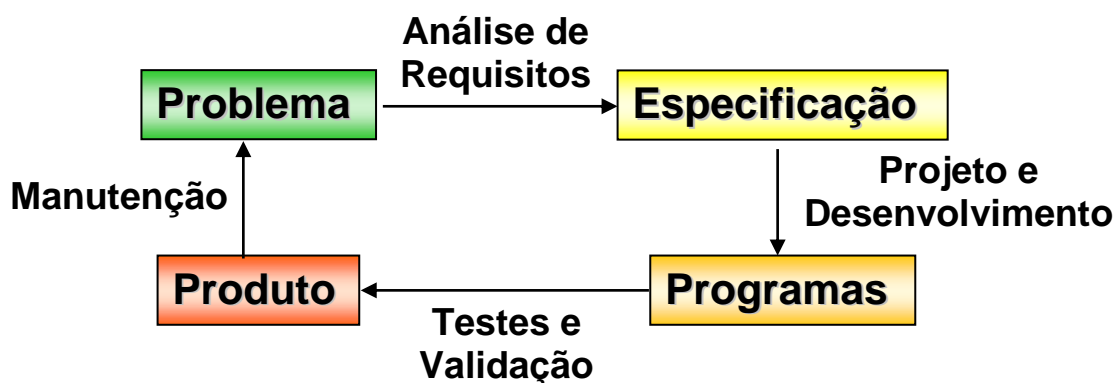
1. Resolução de Problemas pelo Computador



- O computador é uma **ferramenta** que permite a realização do processamento de dados.
- Passos para resolução de problemas:
 - Entendimento do Problema
 - Criação de uma seqüência de operações para solução do problema
 - Execução desta seqüência
 - Verificação da adequação da solução
- O computador desempenha **apenas uma parte** deste processo (3º passo).

2. Fases de Desenvolvimento de Sistemas

O processo de desenvolvimento de sistemas de programação é dividido em 4 fases:



2.1. Análise e Especificação de Requisitos

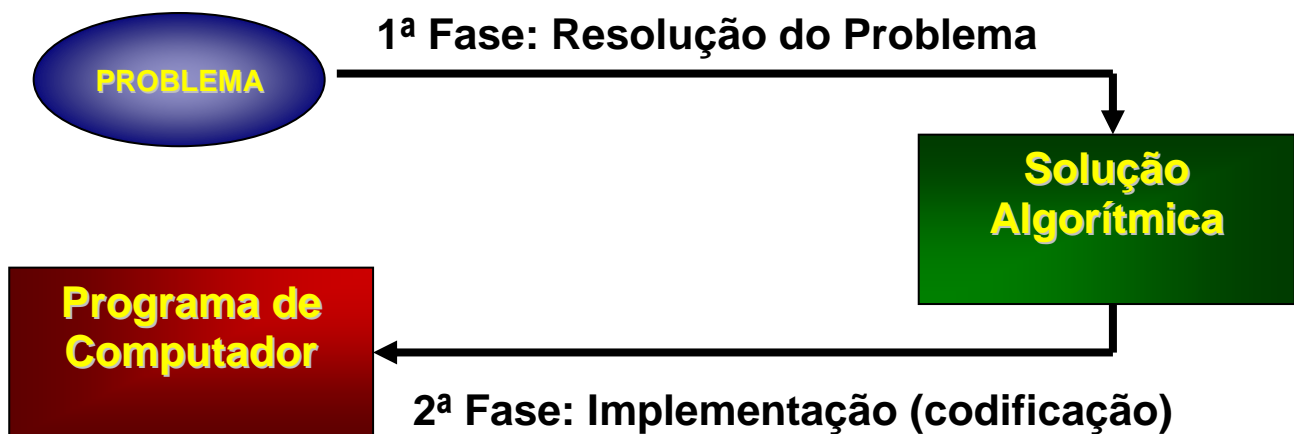
- Um sistema de programação deve satisfazer as necessidades de seus usuários, as quais são expressas na forma de **requisitos**.

Requisito = ação que deve ser executada pelo sistema. (*Ex: registrar as notas dos alunos, calcular a média final, etc.*)

- O levantamento destes requisitos e o seu refinamento (detalhamento) devem ser realizados junto com o usuário e registrado em um documento.
- O sucesso do sistema depende de 3 fatores:
 - Quão bem o sistema captou os requisitos expressos;
 - Quão bem os requisitos captaram as necessidades;
 - Quão bem as necessidades refletem a realidade.

2.2. Projeto e Desenvolvimento do Sistema

- A partir do documento de análise de requisitos, projeta-se o sistema de programação:



- Este processo é dividido em 3 etapas:
 - **Projeto Preliminar:** definição da estrutura modular do software, as interfaces e as estruturas de dados utilizadas;
 - **Projeto Detalhado:** descrição detalhada de cada módulo definido no projeto preliminar (*algoritmo*);
 - **Codificação:** migração das instruções do algoritmo para uma linguagem de programação previamente definida (*programas*).

2.3. Teste e Validação

- Tem por objetivo garantir que o sistema satisfaça os requisitos expressos.
- Consiste da realização de alguns tipos de testes com o intuito de encontrar erros. A inexistência de erros não representa a *adequação operacional* do sistema.
 - **Teste de módulo:** é feito para garantir que o módulo atenda às funcionalidades previstas e às especificações de interface;

- **Teste de integração:** é feito em uma agregação parcial de módulos e visa a detecção das inconsistências nas interfaces entre módulos;
- **Teste de sistema:** é efetuado durante a fase final de validação para assegurar que o sistema funcione de acordo com os requisitos;
- **Teste de instalação:** é realizado durante a instalação do sistema em seu ambiente real de operação, com o objetivo básico de verificar o seu funcionamento neste novo ambiente e corrigir possíveis falhas de instalação;
- **Teste de validação:** é feito junto ao usuário, o qual deve validar o perfeito funcionamento do sistema no seu ambiente real de operação, segundo os requisitos especificados e documentados na 1ª fase.

2.4. Manutenção

- Engloba qualquer alteração no sistema que se fizer necessária após a entrega do sistema.
- Tipos de Manutenção:
 - **Corretiva:** visa a correção de erros/falhas;
 - **Incremental:** visa a inclusão de novas funcionalidades e/ou a alteração dos requisitos originais.
- Um **sistema de boa qualidade** favorece as atividades de manutenção e, conseqüentemente, minimiza os custos despendidos nesta etapa.

**Sistema de Boa
Qualidade**

- Funciona corretamente
- Possui uma boa documentação de todas as etapas de desenvolvimento

3. Algoritmo

- Algoritmo é uma seqüência finita e bem definida de passos que, quando executados, realizam uma tarefa específica ou resolvem um problema.
Ex: Receitas de culinária, manual de instruções, coreografia, etc.
- **Propriedades do algoritmo:**
 - Composto por ações simples e bem definidas (não pode haver ambigüidade, ou seja, cada instrução representa uma ação que deve ser **entendida e realizada**).
 - Seqüência ordenada de ações
 - Conjunto finito de passos

Pergunta: Como saber se já temos detalhes suficientes para o algoritmo ser entendido e realizado?

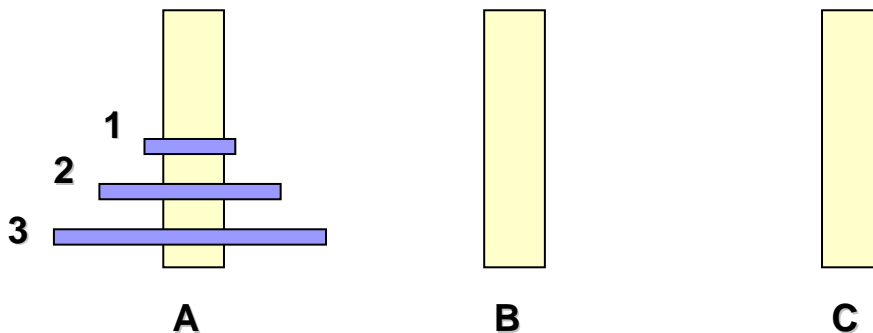
R: Depende da relação de instruções reconhecidas pelo **AGENTE EXECUTOR** do algoritmo.

Ex: receita de bolo \Rightarrow Ser Humano

algoritmo computacional \Rightarrow Computador

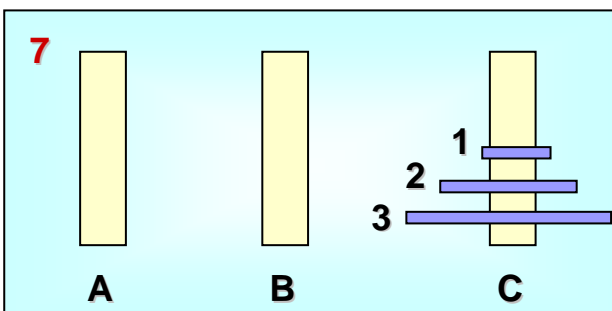
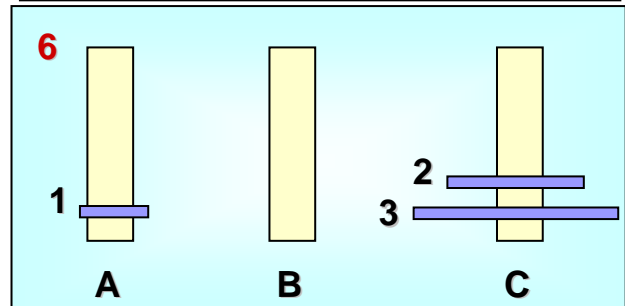
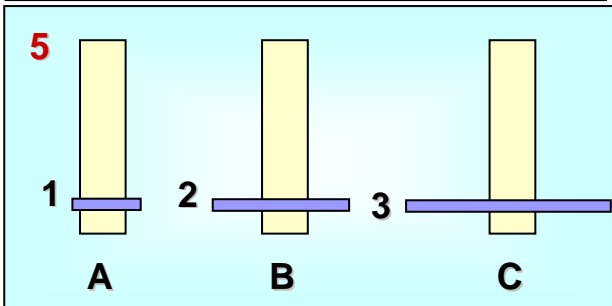
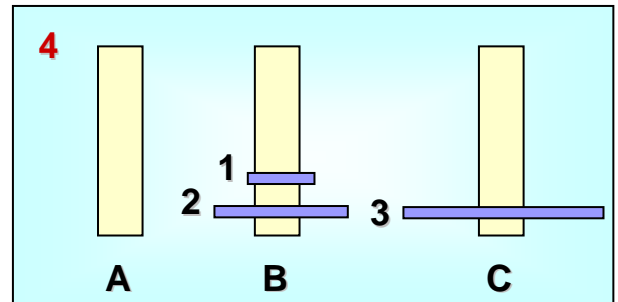
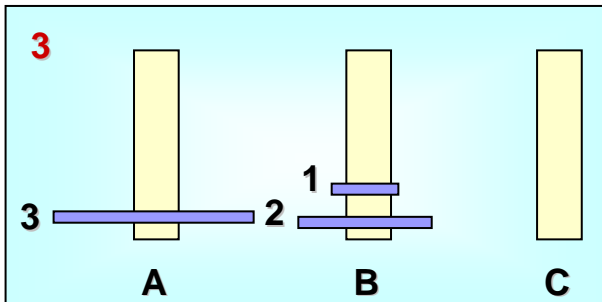
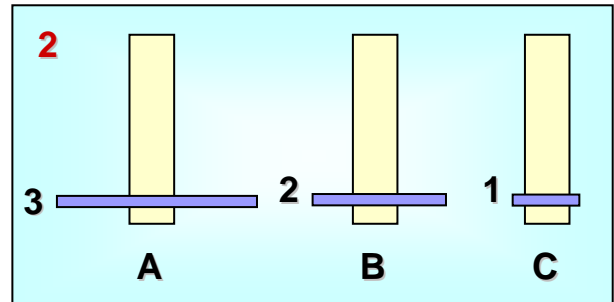
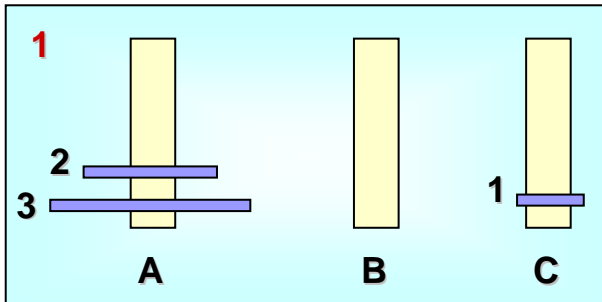
■ **Construindo um Algoritmo** (Problema das Torres de Hanói):

Regra: Mover os discos de uma haste para outra sem que o disco maior fique sobre o disco menor.



Solução:

1. Move o disco 1 para a haste C
2. Move o disco 2 para a haste B
3. Move o disco 1 para a haste B
4. Move o disco 3 para a haste C
5. Move o disco 1 para a haste A
6. Move o disco 2 para a haste C
7. Move o disco 1 para a haste C



Exercícios de Lógica:

1. Temos 3 recipientes de tamanhos distintos (8, 5 e 3 litros), sendo que o recipiente de 8 litros está totalmente cheio. Considerando que os recipientes não sejam graduados, deseja-se colocar 4 litros em dois recipientes.
2. Numa noite, acontece uma queda de energia. Sabendo-se que tem uma vela na gaveta da cozinha, um lampião embaixo da cama, fusíveis de reserva no armário da sala e fósforos na estante da cozinha; descreva a seqüência de ações realizados para diagnosticar e resolver o problema. Neste exercício devemos considerar as seguintes possibilidades:

➤ Fusível queimado; e

- Problema na companhia elétrica.

3. Um comerciante está transportando um lobo, um coelho e 500 kg de cenouras. Durante a viagem, ele se depara com um rio e um pequeno barco, no qual só é possível transportar um elemento por vez. Descreva quais serão as ações tomadas pelo comerciante para atravessar o rio, de modo que ele nunca deixe o lobo e o coelho ou o coelho e as cenouras sozinhos em uma das margens.

4. Algoritmos Computacionais

- Diferem dos algoritmos gerais por serem **executados pelo computador**.
- Diferem dos programas por serem *desenvolvidos em linguagens **NÃO** reconhecidas pelo computador*.
- Auxiliam o usuário na concepção da solução de um problema, **independentemente da linguagem de programação** que será utilizada.
- **Limitações:**
 - Os algoritmos computacionais devem ser expressos nos termos do **conjunto de instruções** entendidas pelo computador.
- **Conceitos básicos** utilizados na construção e interpretação de algoritmos:
 - **Estrutura de Dados:** para manipulação das informações utilizadas no algoritmo.
 - **Estrutura de Controle:** para manipulação das ações.

4.1. Diretrizes para a Elaboração de Algoritmos

- **Identificação do Problema:** determinar o que se quer resolver ou qual objetivo a ser atingido;
- **Identificação das "entradas do sistema":** quais informações estarão disponíveis (serão fornecidas);
- **Identificação das "saídas do sistema":** quais informações deverão ser geradas/calculadas como resultado;
- **Definir os passos a serem realizados:** determinar a seqüências de ações que leve à solução do problema (transforme as entradas nas saídas):
 - Identificar as **regras e limitações do problema**;
 - Identificar as **limitações do computador**;
 - Determinar as **ações possíveis** de serem realizadas pelo computador.
- **Concepção do algoritmo:** registrar a seqüência de comandos, utilizando uma das formas de representação de algoritmos.

- **Teste da solução:** execução manual de cada passo do algoritmo, seguindo o fluxo estabelecido, para detectar possíveis erros.

Exemplo: Calcular a média final dos alunos, sendo que foram realizadas 4 provas de mesmo peso no período.

- Quais os dados de entrada? **P1, P2, P3 e P4**
- Quais os dados de saída? **Média**
- Quais os passos a serem realizados? **Obter notas, calcular média (somar notas e dividir o resultado por 4) e apresentar média**

4.2. Formas de Representação de Algoritmos

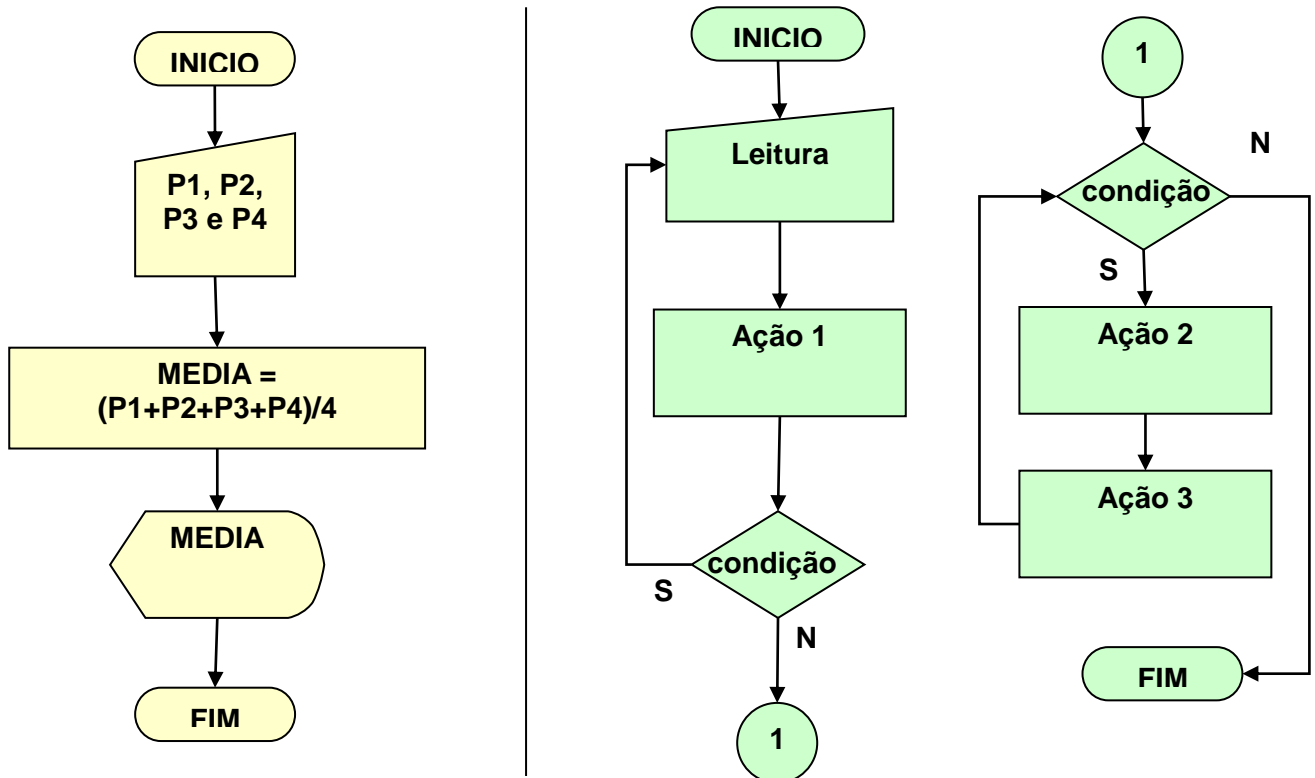
- A descrição de um algoritmo de **forma clara e fácil de ser seguida** ajuda no seu desenvolvimento, depuração (localização e correção de erros) e futura migração para uma linguagem de programação.
- Para facilitar este trabalho, são utilizadas ferramentas específicas de representação da lógica de programação (*seqüência de ações a serem realizadas*).

I. Descrição Narrativa

- Especificação verbal dos passos em linguagem natural.
- Desvantagens:
 - A linguagem natural é imprecisa (possibilita ambigüidades).
 - Proporciona maior trabalho na codificação.
- Sugere-se sua utilização apenas para **comentar** algoritmos e/ou programas, esclarecendo ou realçando pontos específicos.

II. Fluxograma

- Uso de ilustrações gráficas para representar as instruções.
- Apresenta a lógica de um algoritmo, enfatizando passos individuais (objetos gráficos) e o fluxo de execução (setas)
- Desvantagens:
 - Fluxogramas detalhados podem obscurecer a estrutura do programa.
 - Permite transferências arbitrárias de controle

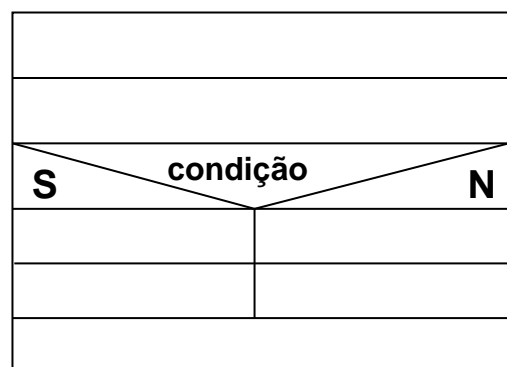


III. Diagramas de Chapin ou Nassi_Shneiderman (Cartas N-S)

- São ferramentas gráficas que representam a função principal de cada passo do algoritmo dentro de uma caixa.

Exemplos:

Leia P1, P2, P3 e P4
Média = $(P1+P2+P3+P4) / 4$
Escreva Média



IV. Pseudolinguagem

- Linguagem especial para desenvolvimento de algoritmos, que **utiliza expressões pré-definidas** para representar ações e fluxos de controle.
- Funciona como uma linguagem simplificada de programação, logo, **facilita a codificação** futura.
- É uma descrição textual, estruturada e regida por regras; que descrevem os passos executados no algoritmo.

- Possui características similares às linguagens de programação:
 - Utiliza palavras-chaves (ex: escreva, se-então, etc.);
 - Identação (alinhamento dos blocos de comandos);
 - Possui um comando por linha;
 - Utiliza ";" como finalizador de comando;
 - Etc.

Obs: será a forma de representação utilizada durante este curso.

Exemplo:

```
algoritmo_Media_Final
inicio
    inteiro P1, P2, P3, P4;
    real media;
    leia(P1, P2, P3, P4);
    media ← (P1+P2+P3+P4)/4;
    escreva(media);
fim
```

5. Estrutura de Dados

- O computador só conhece os **valores que estão armazenados na sua memória**.
- Portanto, a maioria das instruções está, de certa forma, associada ao processo de **armazenamento ou transformação destes valores**.
- Na concepção de algoritmo, pode-se considerar:
 - Memória = conjunto de posições;
 - Cada posição recebe uma identificação (nome) e armazena um valor.

Abstração do Conceito de Memória

IDENTIFICADOR	IDADE	NOME	X1
VALOR	18	"João"	2.5

- As posições de memória **sempre** armazenam um valor e seguem as seguintes premissas:
 - Se armazenamos um novo valor em uma posição, o seu valor antigo será perdido;

- Se nenhum valor for atribuído a uma determinada posição, esta possui um "**lixo**" (as posições nunca estão vazias)

■ **Identificador:**

- Nome de uma posição da memória;
- É definido pelo programador;
- Recomenda-se o uso de nomes significativos.

Exemplo: IDADE, NOME, VLR_SALARIO

Contra-exemplo: X1, KCP, VAR_1, VAR_2

- Regras para definição de identificadores:
 - Deve começar com uma letra;
 - Pode conter letras, números e o caracter especial "_".

Exemplo: NOME, VLR_SALARIO, NOTA_1

Contra-exemplo: 1ª NOTA, C&A, X-1

- **Constantes:** representam valores que não mudam no decorrer do algoritmo

Exemplo: "O nome é:", 24, PI, etc.

- **Variáveis:** representam dados cujos valores são modificados ao longo da execução do algoritmo

5.1. Tipos Primitivos de Dados

■ **Numéricos:**

- Inteiro (ex: 1, -5, 100);
- Real (ex: 1.3, -3.5, 0.55).

■ **Não-Numéricos:**

- Booleano (lógico – ex: True ou False);
- Caracter (alfanumérico – ex: 'A', '@', '1').

Neste tipo, utilizamos ' ' como delimitadores de conteúdo.

5.2. Tipos Estruturados de Dados

■ **Estrutura de Dados Homogênea:**

- **Vetores:** estrutura que suporta ***N posições*** de um mesmo tipo de dado, cada uma com seu respectivo valor.

Ex: *char a[100] /* vetor de 100 caracteres, ou uma string */*

- **Matrizes:** estrutura que suporta ***NxM posições*** de um mesmo tipo de dado, cada uma com seu respectivo valor.

Ex: *int a[10,10] /* matriz de números inteiros 10x10 */*

6. Estrutura Geral de um Algoritmo

algoritmo_NomeAlgoritmo

inicio

Declaração de variáveis
e constantes globais

Blocos de Comandos

fim

- Declaração de variáveis e constantes locais
- Comentários
- Comandos de E/S
- Comandos de Atribuição
- Estruturas de seleção
- Estruturas de repetição
- Chamada de Funções ou Procedimentos

7. Comandos Básicos

7.1. Declaração de Variáveis:

- Define os locais na memória que serão rotulados com o identificador da variável que será utilizada para a manipulação de um determinado tipo de dado.
- Nenhuma variável pode ser usada sem antes ter sido declarada.

Sintaxe: **<tipo_do_dado> Lista_Variaveis;**

Tipo_do_dado = tipos primitivos (inteiro, real, caracter e lógico) ou tipos estruturados (vetores, matrizes e registros).

Lista_Variaveis = conjunto de identificadores.

7.2. Comentários:

- Na verdade, comentários não são comandos, mas são muito importantes para a documentação e entendimento dos algoritmos e programas de computador.
- São compostos por textos explicativos em linguagem natural delimitados entre */** e **/* (ou *//* caso o comentário seja apenas uma linha de texto)

Exemplo: */** Este algoritmo calcula a média

aritmética das notas do semestre **/*

ou

// Este algoritmo calcula a média aritmética das notas do semestre

7.3. Comandos de Entrada e Saída:

- São utilizados para obter (entrada) ou fornecer (saída) informações ao usuário durante a execução do algoritmo.

Sintaxe: **leia(identificador, identificador, ...);**

escreva(termo, termo, ...); *para saída no monitor*

identificador = nome da variável que receberá a informação

termo = uma variável, uma constante ou uma expressão

Exemplo: **leia**(P1, P2, P3, P4);
escreva("A média final é:", media);

7.4. Comando de Atribuição:

- Similar ao sinal de igual das expressões matemáticas convencionais, este comando atribui à variável da esquerda o valor da expressão da direita.

Sintaxe: **variável ← expressão;**

***Variável** = identificador da variável que receberá o valor.*

***Expressão** = pode ser uma variável, uma constante, uma expressão/função matemática ou um expressão lógica.*

Exemplo: nome ← "fulano";
condicao ← **not**(A) **and** B;
sal_liq ← sal_bruto – descontos;
valor_absoluto ← **abs**(valor);

■ Tipos de Operadores:

- **Relacionais:** = (igual), <> (diferente), > (maior que), < (menor que), >= (maior ou igual) e <= (menor ou igual)
- **Lógicos:** **and**, **or**, **not**
- **Aritméticos:** + (soma), - (subtração), * (multiplicação), / (divisão)

Obs: o uso de operadores lógicos e relacionais resultam em **valores lógicos/booleanos (true ou false)**.

■ Algumas Funções:

- **abs(X):** obtém o valor absoluto de X;
- **sqrt(X):** calcula a raiz quadrada de X;
- **log(X):** calcula o logaritmo de X;
- **mod(X,Y):** obtém o resto da divisão de X por Y;
- **trunca(X):** obtém a parte inteira de X;
- **round(X):** arredonda o valor de X;
- **sen(X):** calcula o valor do seno de X;
- **cos(X):** calcula o valor do cosseno de X;
- **tan(X):** calcula o valor da tangente de X.

Precedência de Operadores e Funções

<i>Operadores</i>	<i>Associatividade</i>
Parênteses e Funções	Direita ⇒ Esquerda
Exponenciação e Radiciação	Esquerda ⇒ Direita
Multiplicação e Divisão	Esquerda ⇒ Direita
Soma e Subtração	Esquerda ⇒ Direita
Operadores Relacionais	Esquerda ⇒ Direita
Operadores Lógicos	Esquerda ⇒ Direita

8. Estruturas de Controle

8.1. Estrutura Seqüencial:

- Conjunto de comandos que são executados numa seqüência linear, de cima para baixo, na mesma ordem que aparecem.

Sintaxe: **comando1;**
comando2;

comandoK;

8.2. Estrutura de Seleção:

- Permite a **escolha de um grupo de ações** para serem executadas de acordo com a aceitação ou não de certas condições.
 - As condições que são testadas são do **tipo lógico** (só podem assumir 2 valores - **Verdadeiro** ou **Falso**).

■ **Comando SE:**

- Executa um bloco de comandos específico se a condição testada for verdadeira e um outro bloco de comandos quando a condição for falsa.

Sintaxe: **se** <condição>

bloco de comandos (condição Verdadeira);

senão

bloco de comandos (condição Falsa);

OBS:

1. a cláusula SENÃO é opcional.

2. o bloco de comando pode ser simples (1 comando) ou composto por K comandos, delimitados pela palavras início e fim.

início

comando1;
comando2;
.....
comandoK;

fim

➤ **Exemplos:**

Se (numero < 0)
 numero ← - (numero);

} **Comando SE SEM a cláusula SENÃO**

se (numero < 0)
 escreva("modulo: ", - numero);

senão
 escreva("modulo: ", numero);

} **Comando SE COM a cláusula SENÃO**

■ **Ninhos de SE:**

➤ Dentro de um comando de seleção podemos colocar qualquer tipo de comando, inclusive outros comandos de seleção.

➤ **Exemplo:**

se <condição1>
 se <condição2>
 bloco de comandos (condição2 Verdadeira)
 senão
 se <condição3>
 bloco de comandos (condição3 verdadeira)
 senão
 bloco de comandos (condição3 falsa)

8.3. Estrutura de Repetição:

- Permite a repetição de um grupo de ações. Existem 2 tipos de estrutura de repetição:

- **Condicional:** a repetição ocorre enquanto a condição lógica testada for verdadeira (comandos **ENQUANTO** e **FAÇA ... ENQUANTO**);
- **Incondicional:** tem um número pré-determinado de repetições (comando **PARA**).

- **Comando ENQUANTO:**

- A condição é testada no **início da repetição**.
- Enquanto a condição for **Verdadeira**, o bloco de comandos é executado.
 - * O bloco de comandos pode ser executado **0 ou + vezes**.
- Pára a execução quando a condição se tornar **Falsa**.

Sintaxe: **enquanto** <condição>

bloco de comandos

- **Exemplo:**

algoritmo_quadrado

início

```
inteiro numero; //declaração de variável
```

```
numero ← 1;
```

```
enquanto (numero <> 0)
```

```
    início
```

```
        leia(numero);
```

```
        escreva("o quadrado de ", numero, " é: ");
```

```
        escreva(numero*numero);
```

```
    fim
```

```
        escreva("FIM DO PROGRAMA");
```

```
fim
```

- **Comando FAÇA ... ENQUANTO:**

- A condição é testada no **final da repetição**.
- Enquanto a condição for **Verdadeira**, o bloco de comandos é executado.
 - * O bloco de comandos é executado pelo menos **1 vezes**.
- Pára a execução quando a condição se tornar **Falsa** (condição de parada).

Sintaxe: **faça**

bloco de comandos

```
enquanto <condição>;
```

➤ **Exemplo:**

algoritmo_Quadrado

inicio

inteiro numero;

Faça

inicio

leia(numero);

escreva("o quadrado de ", numero, " é: ");

escreva(numero*numero);

fim

enquanto (numero <> 0);

escreva("FIM DO PROGRAMA");

fim

■ **Comando PARA:**

- Repete o bloco de comandos enquanto a variável de controle for **menor ou igual** ao valor final (vlr_fim).
- A variável de controle recebe um valor inicial (vlr_ini) e é **incrementada automaticamente** pelo parâmetro de incremento padrão (acrécimo de 1), quando a cláusula **PASSO** é omitada, ou pelo valor definido pelo usuário através desta cláusula.
- A variável de controle **NÃO** pode ser modificada no bloco de comandos.

Sintaxe: **para** <var_controle> = vlr_ini **ate** vlr_fim **passo** <inc> **faça**
bloco de comandos;

➤ **Exemplo:**

Algoritmo_SomaQuadrados

inicio

inteiro S2;

S2 = 0;

para cont = 1 **ate** 10 **passo** 1 **faça**

inicio

s2 ← s2 + cont * cont;

fim

escreva("Soma dos quadrados dos 10 primeiros números: ", s2);

fim

9. Outras Informações

9.1. Importância das Variáveis

- As variáveis representam as informações manipuladas pelo algoritmo:

TIPO	EXEMPLO
Dados fornecidos pelo usuário	leia(nome , idade);
Resultados de expressões	area ← base * altura;
Acumuladores/Contadores	cont ← cont + 1; /* Conta */ soma ← soma + num; /*Acumula*/
Finalizadores de repetições	numero = 0
Sinalizadores/Flags	atualizado ← "N"; /* Não Lógico */ atualizado ← False; /* Lógico */

9.2. Dicas na Confecção de Algoritmos

- Para construir um algoritmo mais claro e legível, deve-se utilizar:
 - **Comentários de cabeçalho** (descreve sucintamente o algoritmo) **e de linha** (descreve a ação realizada por uma determinada linha de comando);
 - **Identificadores representativos**;
 - **Identação** das estruturas lógicas do algoritmo;
 - **Linhas em branco entre estruturas** lógicas complexas (ex: declaração de variáveis, ninhos de SE, estruturas de controle, etc.)

9.3. Teste de Mesa

- Consiste do acompanhamento manual (linha a linha) da execução do algoritmo, visando:
 - Avaliar se os resultados obtidos corresponde àqueles esperados/desejados.
 - Detectar, se existentes, os erros de comandos e/ou fluxo de execução.
- Durante os testes, deve-se definir o valores de entrada, visando avaliar as seguintes situações:
 - Casos extremos (*valores limítrofes da validade*);
 - Exceções do problema (*valores inválidos*).

■ **Exemplo:**

algoritmo_Obtem_Conceito_Aluno

inicio

int P1, P2, P3, P4;

real media;

leia(P1,P2,P3,P4);

media ← (P1+P2+P3+P4) / 4;

escreva(media);

se media < 6

 escreva("REPROVADO");

senao

 escreva("APROVADO");

fim

Valores das Entradas: P1 = 5, P2 = 6, P3 = 8 e P4 = 4

Teste1	P1	P2	P3	P4	media	media < 6
<i>Declare</i>	-	-	-	-	-	-
<i>int: P1,P2,P3,P4;</i>	*	*	*	*	-	-
<i>real: media;</i>	*	*	*	*	*	-
<i>Inicio</i>	*	*	*	*	*	-
<i>leia(P1,P2,P3,P4);</i>	5	6	8	4	*	-
<i>media ← (P1+P2+P3+P4) / 4;</i>	5	6	8	4	5,75	-
<i>escreva(media);</i>	5	6	8	4	5,75	-
<i>se media < 6 entao</i>	5	6	8	4	5,75	true
<i>Demais comandos</i>	5	6	8	4	5,75	true

* Representa um valor não conhecido (lixo)

Valores das Entradas: P1 = -5, P2 = 11, P3 =12 e P4 = 15

Teste2	P1	P2	P3	P4	media	media < 6
<i>Declare</i>	-	-	-	-	-	-
<i>int: P1,P2,P3,P4;</i>	*	*	*	*	-	-
<i>real: media;</i>	*	*	*	*	*	-
<i>Inicio</i>	*	*	*	*	*	-
<i>leia(P1,P2,P3,P4);</i>	-5	11	12	15	*	-
<i>media ← (P1+P2+P3+P4) / 4;</i>	-5	11	12	15	8,25	-
<i>escreva(media);</i>	-5	11	12	15	8,25	-
<i>se media < 6 entao</i>	-5	11	12	15	8,25	False
<i>Demais comandos</i>	-5	11	12	15	8,25	false

- **Este teste identifica duas falhas no algoritmo:** *aceitar valor negativo e valores maiores que 10.*

1. **Exercícios de Fixação:** Teste os algoritmos abaixo:

<pre> algoritmo_ExecA inicio real A, B; inteiro C, X; A ← 6.0; B ← A/2; C ← 11; X ← trunca(C / 4); C ← mod(C, 2); B ← 5.4; C ← C+1; A ← B+2; fim </pre>	<pre> algoritmo_ExecB inicio inteiro C, X, R; leia(C, X); se C > X entao inicio escreva(0); fim senao inicio faça inicio R ← R + C; C ← C+1; fim enquanto (C > X); escreva(R); fim fim </pre>
---	---

<pre> algoritmo_ExecC inicio inteiro c, num, exp, res; leia(num, exp); res ← 1; para c=1 ate exp passo 1faça inicio res ← res * num; fim escreva(res); fim </pre>	<pre> algoritmo_ExecD inicio inteiro A, B, C; real Delta, X1, X2; leia(A, B, C); Delta ← B*B - 4*A*C; se (Delta < 0) inicio escreva("Não existem raízes reais!"); fim senão inicio X1 = (-B + sqrt(Delta)) / 2*A; X2 = (-B - sqrt(Delta)) / 2*A; escreva(X1, X2); fim fim </pre>
---	---

10. Programação Estruturada (Modularização)

- A característica fundamental da programação estruturada é **modular a resolução do problema** através da sua divisão **em subproblemas menores e mais simples**.
 - Neste processo, cada subproblema pode ser **analisado de forma individual e independente** dos demais.
- Tal modularização tem por objetivo:
 - Facilitar o **trabalho com problemas complexos**;
 - Permitir a **reutilização de módulos**
- **Refinamentos Sucessivos (top-down)**: Essa técnica consiste da divisão do problema inicial em subproblemas, e estes em partes ainda menores, sucessivamente, até que **cada parte seja descrita através de um algoritmo claro e bem-definido**.
 - Um algoritmo que resolve um determinado subproblema é denominado **subalgoritmo**.

10.1. Subalgoritmos (Módulos)

- Por convenção, um subalgoritmo deve ser **declarado acima dos módulos que o chamam**.
- Todo subalgoritmo tem por objetivo a resolução de um determinado subproblema. Portanto, mantém **as mesmas características de um algoritmo comum**, ou

seja, pode ter dados de entrada; dados de Saída; e conter **qualquer tipo de comando** aceito por um algoritmo.

- Por serem tratados de forma independente dos demais módulos, cada subalgoritmo só pode manipular variáveis/constantes:
 - **Globais:** declaradas no início do algoritmo principal (*passagem de valor por referência*); ou
 - **Locais:** declaradas no próprio subalgoritmo (*passagem de valor por argumento*).
- Subalgoritmos podem ser do tipo **Procedimento** (não retorna valor de forma explícita) ou **Função** (retorna valor de forma explícita).

10.2. Troca de Informação

- Existem duas formas de interagir com subalgoritmos:
 - **Explícita:** quando a troca é feita **através de argumentos**. Existem 2 modos de passar argumentos para um subalgoritmo:
 - ★ **Por valor:** o argumento formal será uma *variável independente*, ou seja, ocupará um espaço na memória durante a execução do módulo. Qualquer alteração no **argumento formal**, não afeta o **argumento atual**.
 - ★ **Por referência:** o **argumento formal** não ocupará espaço de memória, pois utilizará o espaço já alocado pelo **argumento atual**. Qualquer alteração realizada no subalgoritmo **afetará o argumento atual**.
 - **Implícita:** quando a troca é feita **pela utilização de variáveis globais**.
 - ★ Esta forma funciona de modo similar à passagem de argumentos por referência, pois **não necessita de nova alocação de memória** e qualquer alteração realizada durante a execução do módulo **afeta o conteúdo da variável para os demais módulos**.

10.3. Argumentos

- Na escrita de um subalgoritmo, os dados necessários para a resolução do subproblema podem ser informados na declaração do módulo. Esses argumentos são chamados de argumentos formais.

Ex: funcao expoente(**BASE**:real, **FATOR**:int)

- Os argumentos usados na chamada do módulo são denominados argumentos atuais.

Ex: resultado ← expoente(**NRO**, **EXP**);

- Tanto a **quantidade**, quanto o **tipo de dado** dos argumentos atuais **devem ser os mesmos** dos respectivos argumentos formais.

10.4. Subalgoritmo Procedimento

- Interface com outros módulos:
 - **Entrada:** pode ser explícita ou implícita;
 - **Saída:** somente implícita.

Sintaxe:

procedimento <nome_proc>(arg1:tipo, arg2:tipo, ... , argN:tipo)

inicio

<declaração de variáveis/constantes locais>

<bloco de comandos>

fim

inicio //função principal

<nome_proc>(var1, var2, ... , varN);

Fim

10.5. Subalgoritmo Função

- Interface com outros módulos:
 - **Entrada:** *pode ser explícita ou implícita;*
 - **Saída:** *sempre explícita (associado ao nome da função), podendo também ter saídas implícitas (não é usual).*

Sintaxe:

tipo_funcao **funcao** <nome_func>(arg1:tipo, arg2:tipo, ... , argN:tipo)

inicio

<declaração de variáveis/constantes locais>

<bloco de comandos>

fim

inicio //função principal

var ← <nome_func>(var1, var2, ... , varN);

fim

10.6. Considerações Finais

■ Exemplo de Utilização de Subalgoritmos

```

algoritmo_exemplo
inteiro num1, num2; //variáveis global
procedimento ler_dados() /* Lê a base e o fator da exponenciação */
inicio //procedimento
    faça
        inicio
            leia(num1, num2);
        fim
    enquanto (num1 <= 0) and (num2 < 0)

fim //procedimento
inteiro funcao exp(base:int, fator:int) /* Faz base ^ fator */
inicio //funcao
    inteiro cont, resultado;
    resultado ← 1;
    para cont =1 ate fator faça
        inicio //para
            resultado ← resultado * base;
        fim //para
    exp ← resultado;
fim //funcao
inicio //função principal
    ler_dados() /* chamada do procedimento */;
    escreva(exp(num1, num2)); /* chamada da função */
fim //função principal
    
```

11. Referências Bibliográficas

- Boratti, I. C. "Introdução à Programação – Algoritmos", 2ª edição, editora Visual Books;
- ASCENCIO, A. F. G. "Fundamentos da programação de computadores", editora Pearson Prentice Hall;
- GUIMARÃES, A. M. "Algoritmo e estrutura de dados", editora LTC;
- FORBELLONE, A. L. V. "Lógica de programação", edit. Pearson Prentice Hall;
- Farrer, C. G. B., Faria, E. C. "Algoritmos Estruturados", 3ª edição, editora LTC;
- Material da internet.