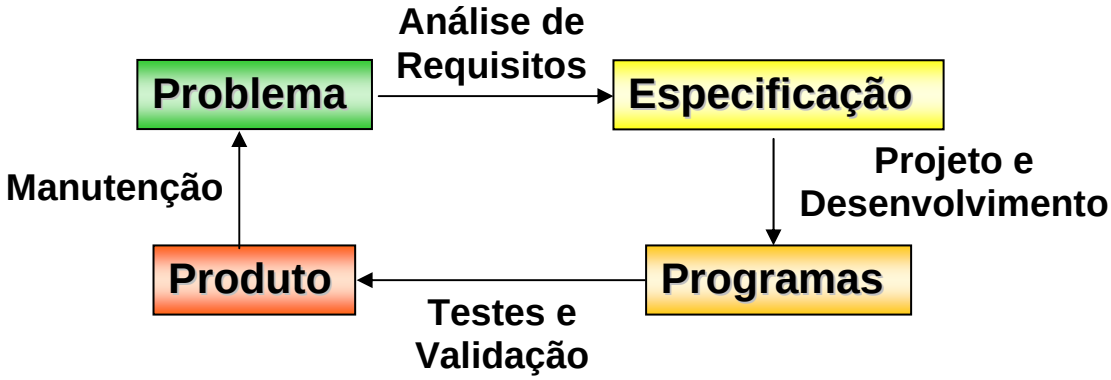
Introdução à Algoritmos

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

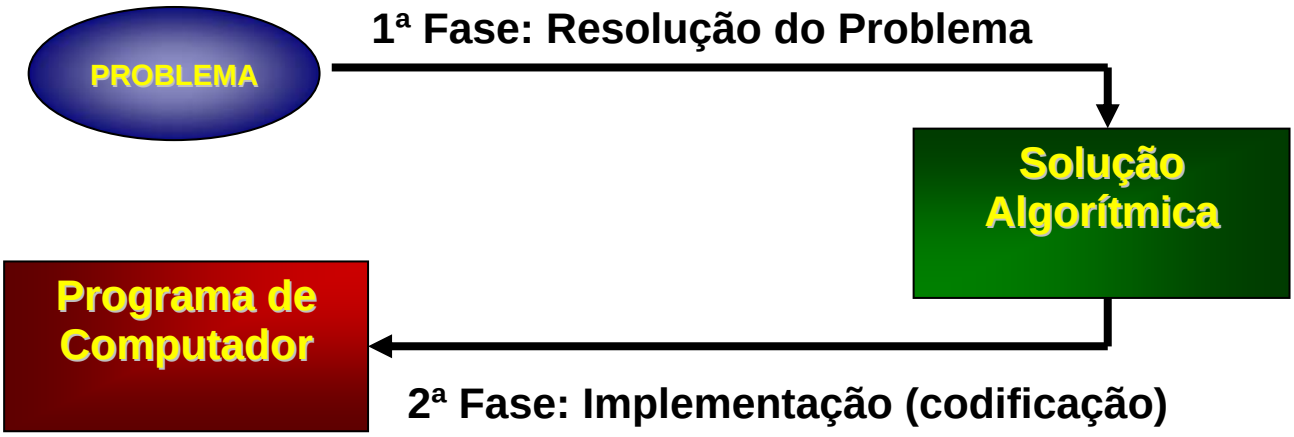
- Um sistema de programação deve satisfazer as necessidades de seus usuários, as quais são expressas na forma de **requisitos**.

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 da 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, consequentemente, 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?

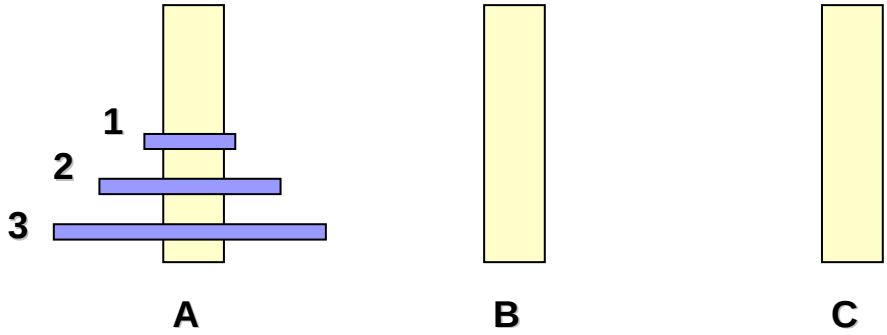
R: Depende da relação de instruções reconhecidas pelo **AGENTE EXECUTOR** do algoritmo.

Ex: receita de bolo ⇒ Ser Humano

algoritmo computacional ⇒ Computador

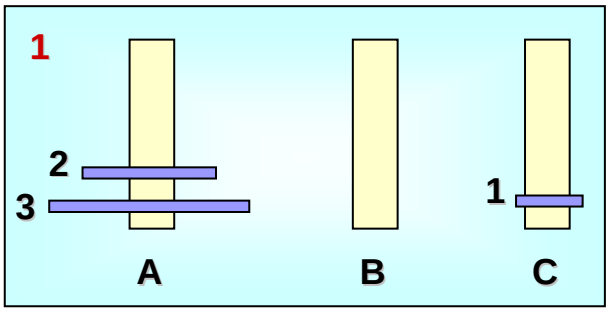
- **Construindo um Algoritmo** (Problema das Torres de Hanói)**:**

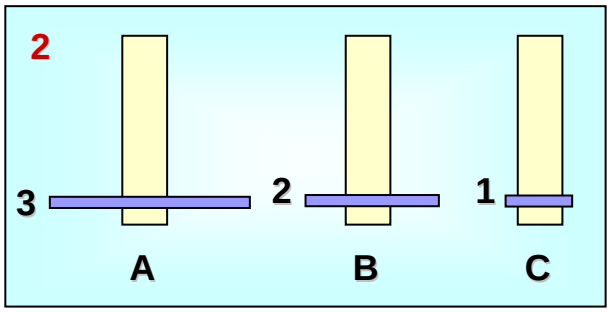
  Regra: Mover os discos de uma haste para outra sem que o disco maior fique sobre o disco menor.

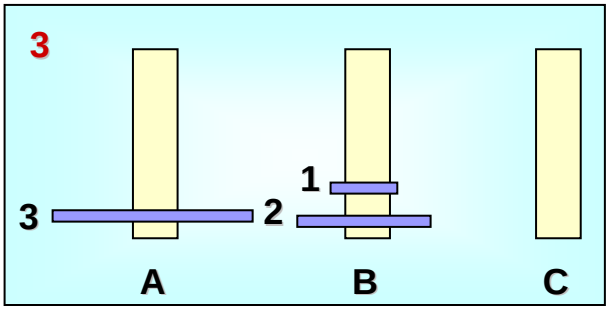


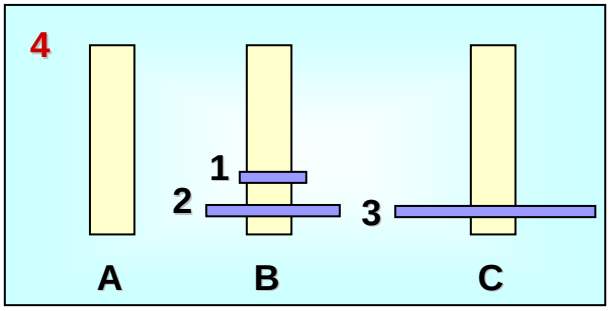
Solução:

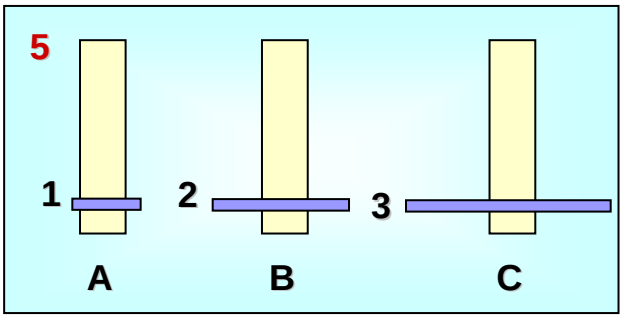
1. Move o disco 1 para a haste C
2. Move o disco 2 para a haste B
3. Move o disco 1 para a haste B
4. Move o disco 3 para a haste C
5. Move o disco 1 para a haste A
6. Move o disco 2 para a haste C
7. Move o disco 1 para a haste C

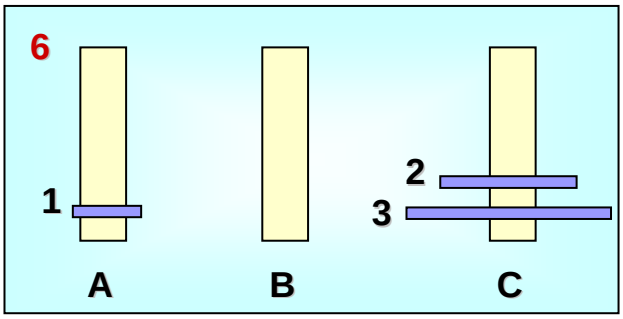


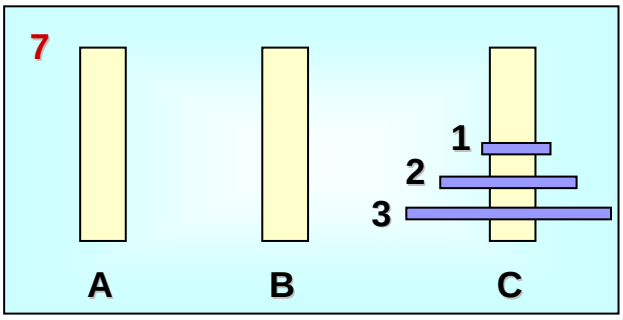












Exercícios de Lógica:

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

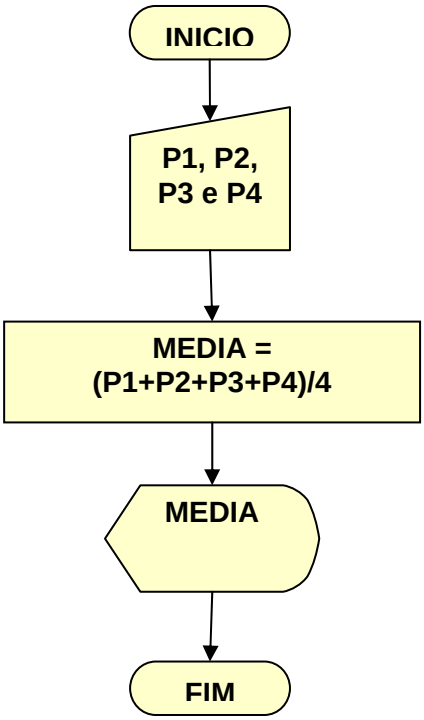
- A descrição de um algoritmo de **forma clara e fácil de ser seguida** ajuda no seu desenvolvimento, depuração (localização e correção de erros) e futura migração para uma linguagem de programação.
- Para facilitar este trabalho, são utilizadas ferramentas específicas de representação da lógica de programação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

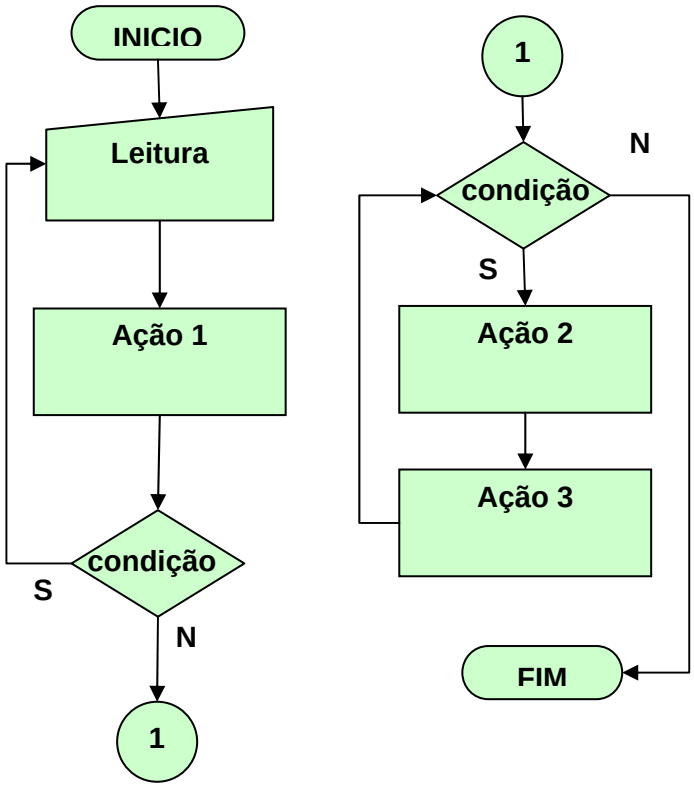
I. Descrição Narrativa

- 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



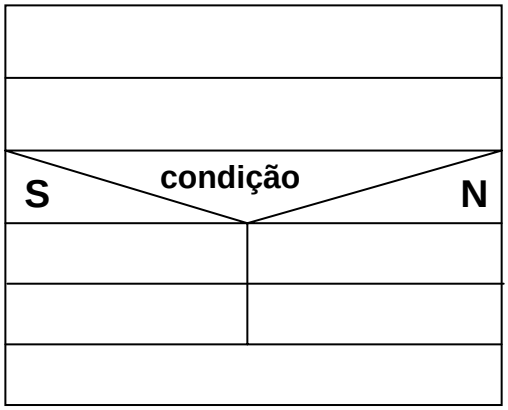


III. Diagramas de Chapin ou Nassi_Shneiderman (Cartas N-S)

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

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”, ‘@’, “x1”).

    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]; /* string – cadeia de até 100 caracteres */*
  - **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 */*
- **Estrutura de Dados Heterogênea** :
  - **Registros**: estrutura que suporta **K variáveis**, cada qual com um tipo de dado próprio.

    EX: *cadastro* ***registro {***

    Char nome[100];

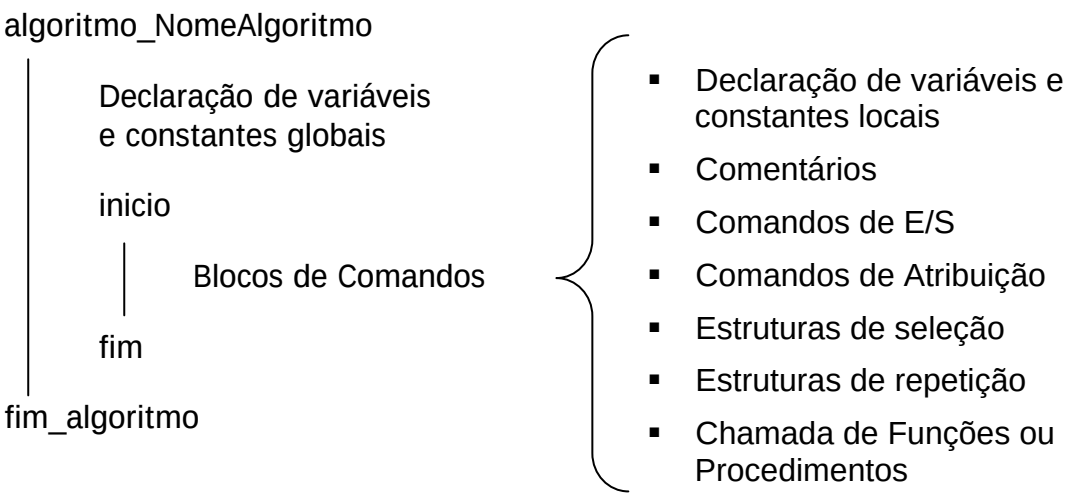
    int idade;

    real salário;

    *booleano ativo***}** *;*

Obs*: os tipos estruturados podem ser combinados de modo a formar estruturas complexas de dados.*

6. Estrutura Geral de um Algoritmo



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: **declare**

<tipo_do_dado> Lista_Variáveis;

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

Lista_Variáveis *= 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 **//** textos)

Exemplo: /* Este algoritmo calcula a média aritmética das notas do semestre ***/**

OBS: Algumas literaturas utilizam { }, mas não empregaremos estes símbolos para não confundir com os delimitadores do C.

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***

imprima(termo, termo, ...); ***para saída na impressora***

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**);**

imprima("A média final é:"**,** (P1+P2+P3+P4)/4**);**

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**;**

cad.sal_liq ← cad.sal_bruto **–** cad.descontos**;**

valor_absoluto ← **abs(**matriz_identidade[2,4]**);**

- **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), % (resto da divisão inteira), e **^** (exponenciação).

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

- **Tipos de 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

Operadores	*Associatividade*
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

Exercícios de Fixação

1. Faça 2 exemplos para cada um dos conceitos a seguir: entrada de dados; saída de dados; declaração de variáveis (tipos primitivos); declaração de variáveis (tipos compostos); e comandos de atribuição de expressões aritméticas e funções.

2. Utilizando tipos primitivos, crie declarações de variáveis que armazenem as seguintes informações: o nome de um objeto geométrico, a quantidade de lados, a área, o perímetro e se o objeto é regular ou não.

3. Refaça o exercício anterior utilizando dados estruturados, com o objetivo de armazenar os dados de até 10 objetos.

4. Escreva um algoritmo que leia 2 números inteiros e retorne no monitor os resultados da soma, subtração, multiplicação, divisão, divisão inteira e resto da divisão destes números.

8. Estruturas de Controle

.Controla a ordem em que os comandos são executados (fluxo de execução).

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; ..... ;

comandoN;

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> **então**

bloco de comandos (condição Verdadeira);

senão

bloco de comandos (condição Falsa);

fim-se

OBS: a cláusula SENÃO é opcional.

- **Exemplos:**

Se numero < 0 **então**

numero ← abs(numero);

fim-se

Comando SE SEM a cláusula SENÃO

```
se cadastro[1].sexo = "F" então
|       escreva("Sra. ", cadastro[1].nome);
senão
|       escreva("Sr. ", cadastro[1].nome);
fim-se
```

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> então
|   se <condição2> então
|   |       bloco de comandos (condição2 Verdadeira);
|   fim-se
senão
|   se <condição3> então
|   |       bloco de comandos (condição3 verdadeira);
|   senão
|   |       bloco de comandos (condição3 falsa);
|   fim-se
fim-se
```

- **Comando CASO:**
  - Testa várias condições para uma mesma variável e executa o bloco de comandos relacionado à condição verdadeira.
  - Substitui os ninhos de comando SE, quando estes são feitos com base em uma **mesma expressão e testarem somente igualdades**.

Sintaxe: **caso** <identificador_variável>

```
|   <op1>:    bloco de comandos;
|   <op2>:    bloco de comandos;
|   <op3>:    bloco de comandos;
```

```
        .........
        <opk>:      bloco de comandos;
        outro:      bloco de comandos;
    fim-caso
```

OBS: A cláusula <u>OUTRO</u> do comando é opcional.

- **Exemplo:**

```
caso estado_civil
        "S":  escreva("Solteiro");
        "C":  escreva("Casado");
        "D":  escreva("Divorciado");
        "V":  escreva("Viúvo");
        outro: escreva("Desconhecido");
fim-caso
```

- **Exercícios de Fixação:**

1. Faça um algoritmo para montar um menu na tela com 5 opções (1 - Cadastro, 2 - Edição, 3 - Consulta, 4 - Exclusão e 5 - Sair), leia a opção escolhida e chame a função correspondente.

   Obs: neste exercício, a chamada de uma função equivale a digitar o seu nome.

1. Elabore um algoritmo que dado um número inteiro qualquer, responda se ele é positivo, negativo ou nulo (igual a zero).

2. Modifique o algoritmo anterior, de modo que ele responda se o número é par ou ímpar.

3. Escreva um algoritmo que leia o nome e a inicial do estado civil (C, S, V, D ou O) de N pessoas, até que seja digitado "FIM" no nome. No final, este algoritmo deve retornar a lista das pessoas cadastradas e seus respectivos estados civis (**C**asado, **S**olteiro, **V**iúvo, **D**ivorciado ou **O**utro).

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 **REPITA**);

- **Incondicional**: tem um número pré-determinado de repetições (comando **PARA**).

- **Comando <u>ENQUANTO</u>:**

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

Sintaxe: **enquanto** <condição> **faça**

bloco de comandos**;**

fim-enquanto

- **Exemplo:**

```
numero ← 1;
enquanto numero ≠ 0 faça
    leia(numero);
    escreva("o quadrado de ", numero, " é: ");
    escreva(numero^2);
fim-enquanto
escreva("FIM DO PROGRAMA");
```

- **Comando <u>FAÇA</u> ... <u>ENQUANTO</u>:**

  - A condição é testada no **final da repetição**.
  - Enquanto a condição for **<u>Verdadeira</u>**, o bloco de comandos é executado.
    - O bloco de comandos é executado pelo menos **1 vez.**
  - Pára a execução quando a condição se tornar **<u>Falsa</u>**.

Sintaxe: **faça**

bloco de comandos**;**

enquanto <condição>;

- **Exemplo:**

```
numero ← 1;
faça
        leia(numero);
        escreva("o quadrado de ", numero, " é: ");
        escreva(numero^2);
enquanto numero ≠ 0;
escreva("FIM DO PROGRAMA");
```

- **Comando <u>REPITA</u>:**
  - A condição é testada no **final da repetição**.
  - Enquanto a condição for **<u>Falsa</u>**, o bloco de comandos é executado.
    - ✶ O bloco de comandos é executado pelo menos **1 vez.**
  - Pára a execução quando a condição se tornar **<u>Verdadeira</u>** (condição de parada).

Sintaxe:

```
repita
        bloco de comandos;
ate <condição>
```

- **Exemplo:**

```
repita
        leia(numero);
        escreva("o quadrado de ", numero, " é: ");
        escreva(numero^2);
ate numero = 0
escreva("FIM DO PROGRAMA");
```

- **Comando <u>PARA</u>:**

  - 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 (acréscimo de 1), quando a cláusula **<u>PASSO</u>** é omitada, ou pelo valor definido pelo usuário através desta cláusula.

  - A variável de controle **<u>NÃO</u>** pode ser modificada no bloco de comandos.

  Sintaxe: **para** <var_controle> = vlr_ini **ate** vlr_fim **passo** <inc> **faça**

  | bloco de comandos**;**

  fim-para

  - **Exemplo:**

```
para cont = 1 ate 9 passo 2 faça
|       vetor[cont] ← cont;
|       vetor[cont+1] ← cont;
fim-para

/* escreve o vetor na ordem inversa */

para cont = 10 ate 1 passo -1 faça
|       escreva(vetor[cont]," , ");
fim-para
```

- **Exercícios de Fixação:**

1. Elabore um algoritmo para determinar o menor número inteiro fornecido pelo usuário. Considere que o número zero indica o fim da entrada de dados.

2. Construa um algoritmo que calcule e imprima a somatória de N números (sendo N > 0). Considere como dados de entrada a quantidade de números a serem lidos e os valores dos respectivos números.

3. Faça um algoritmo que, sem utilizar o operador de exponenciação, realize a operação XY, para qualquer X e Y fornecido pelo usuário.

4. Reescreva o algoritmo acima, utilizando as demais estruturas de repetição.

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	ate **resp** = "N"
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 correspondem àqueles esperados/desejados.
  - Detectar, se existentes, os erros de comandos e/ou fluxo de execução.
- Durante os testes, deve-se definir os valores de entrada, visando avaliar as seguintes situações:
  - Casos extremos (*valores limítrofes da validade*);
  - Exceções do problema (*valores inválidos*).
- **Exemplo:**

```
algoritmo_Obtem_Conceito_Aluno
    declare
        int  P1, P2, P3, P4;
        real  media;
    inicio
        leia(P1,P2,P3,P4);
        media ← (P1+P2+P3+P4) / 4;
        escreva(media);
        se media < 6 entao
            escreva("REPROVADO");
        senao
            escreva("APROVADO");
        fim-se
    fim
fim-algoritmo
```

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

Teste1	P1	P2	P3	P4	media	media < 6
Declare	-	-	-	-	-	-
int P1,P2,P3,P4;	*	*	*	*	-	-
real media;	*	*	*	*	*	-
Inicio	*	*	*	*	*	-
leia(P1,P2,P3,P4);	5	6	8	4	*	-
media ← (P1+P2+P3+P4) / 4;	5	6	8	4	5,75	-
escreva(media);	5	6	8	4	5,75	-
se media < 6 entao	5	6	8	4	5,75	true
Demais comandos	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
Declare	-	-	-	-	-	-
int P1,P2,P3,P4;	*	*	*	*	-	-
real media;	*	*	*	*	*	-
Inicio	*	*	*	*	*	-
leia(P1,P2,P3,P4);	-5	11	12	15	*	-
media ← (P1+P2+P3+P4) / 4;	-5	11	12	15	8,25	-
escreva(media);	-5	11	12	15	8,25	-
se media < 6 entao	-5	11	12	15	8,25	False
Demais comandos	-5	11	12	15	8,25	false

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

■ Exercícios de Fixação:

1. Faça o teste de mesa no algoritmo do exercício 4 do slide 50 e verifique se é necessária a inclusão de alguma consistência de dados.

2. Teste os algoritmos abaixo:

algoritmo_Exec2.a

```
        declare
                real   A, B;
                int   C, X;
        inicio
                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
fim-algoritmo
```

algoritmo_Exec2.b

```
        declare
                int   c, num, exp, res;
        inicio
                leia(num, exp);
                res ← 1;
                para c=1 ate exp faça
                        res ← res * num;
                fim-para
                escreva(res);
        fim
fim-algoritmo
```

algoritmo_Exec2.c

```
        declare
                int   C, X, R;
        inicio
                leia(C, X);
                se C > X entao
                        escreva(0);
                senao
                        repita
                                R ← R + C;
                                C ← C+1;
                        ate C > X
                        escreva(R);
                fim-se
        fim
fim-algoritmo
```

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).

10.1.1. 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*** n*ão 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 modulo **afeta o conteúdo da variável para os demais módulos**.

10.1.2. 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.

10.1.3. 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**)**

    <declaração de variáveis/constantes locais>

    inicio

        <bloco de comandos>

    fim

fim-procedimento

inicio

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

fim

10.1.4. 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:

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

    <declaração de variáveis/constantes locais>

    inicio

        <bloco de comandos>

    fim

fim-funcao

inicio

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

fim

10.1.5. Considerações Finais

- **Exemplo de Utilização de Subalgoritmos**

```
algoritmo_exemplo
    Declare
        Int  num1, num2;
    procedimento ler_dados() /* Lê a base e o fator da exponenciação */
        inicio
            repita
                leia(num1, num2);
            ate (num1 > 0) and (num2 >= 0)
        fim
    fim-procedimento
    funcao exp(base:int, fator:int): int    /* Faz base ^ fator */
        declare
            int  cont, resultado;
        inicio
            resultado ¬ 1;
            para cont=1 ate fator faca
                resultado ¬ resultado * base;
            fim-para
            exp ¬ resultado;
        fim
    fim-funcao
    inicio
        ler_dados() /* chamada do procedimento */;
        escreva(exp(num1, num2)); /* chamada da função */
    fim
fim-algoritmo
```

- **Exercícios de Fixação:**

1. Utilizando os conceitos de procedimentos e funções, construa um algoritmo que determine, para uma série de valores inteiros, quantos são maiores que a média do conjunto.

2. Utilizando os conceitos de procedimentos e funções, desenvolva um algoritmo que calcule, a partir de 2 números inteiros ***P*** e ***M***, a combinação de ***M*** elementos tomados ***P*** a ***P***, conforme a expressão:

$$C_M^P = \frac{M!}{P!(M-P)!}$$

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.