BCC 201 - Introdução à Programação

Conceitos Básicos de Programação

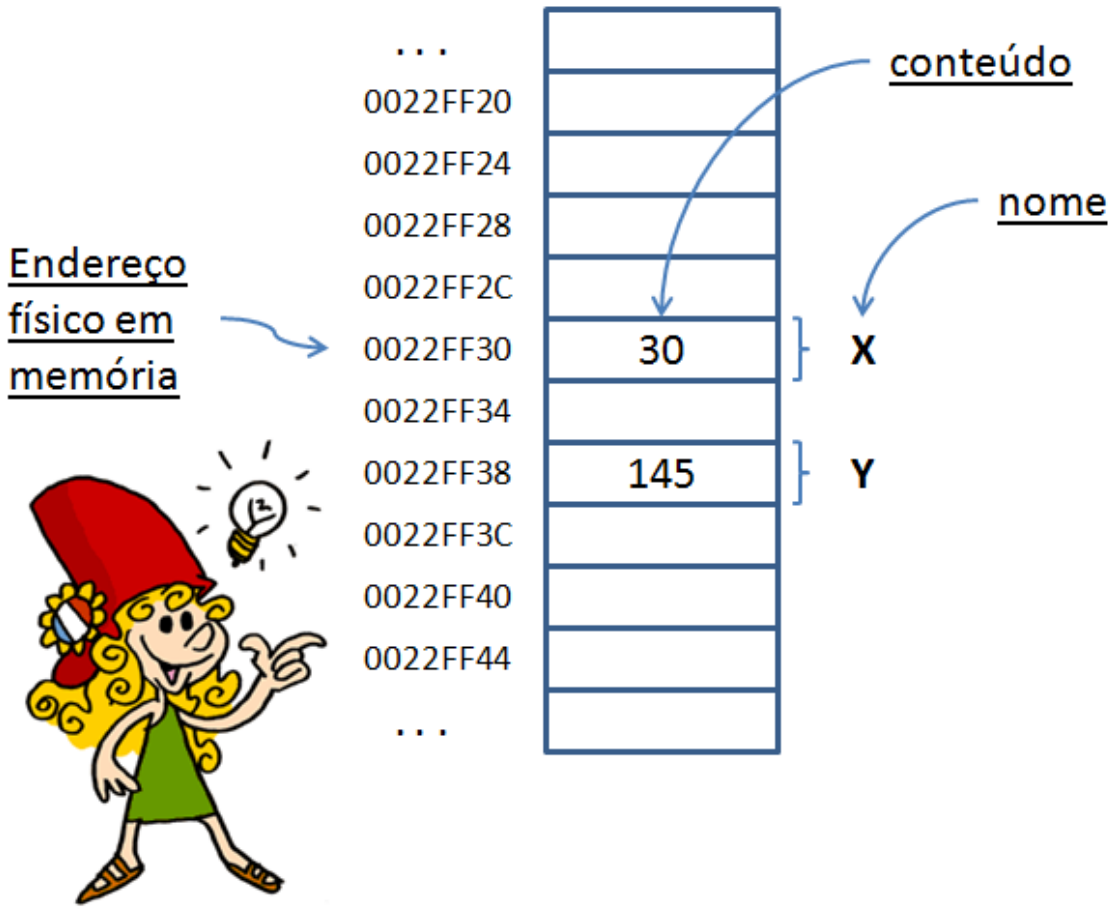
Guillermo Cámara-Chávez
UFOP

Conceitos básicos I

- **Variável**

Conceitos básicos II

- **Posição de memoria**, **identificada** através de um **nome**, e usada para **guardar um valor**



Conceitos básicos III

- Programador acessa variáveis através de algoritmos visando atingir resultados propostos

Conceitos básicos IV

- Existe a partir da declaração e seu escopo pode ser uma rotina ou todo o programa (locais $\times$ globais)

global (todos acessam)

local(só pertence a ele)

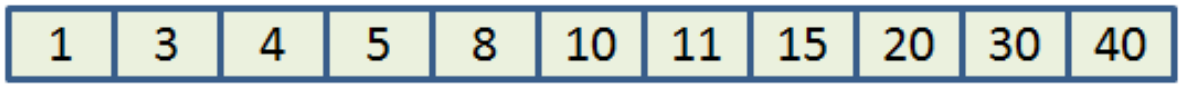
Conceitos básicos V

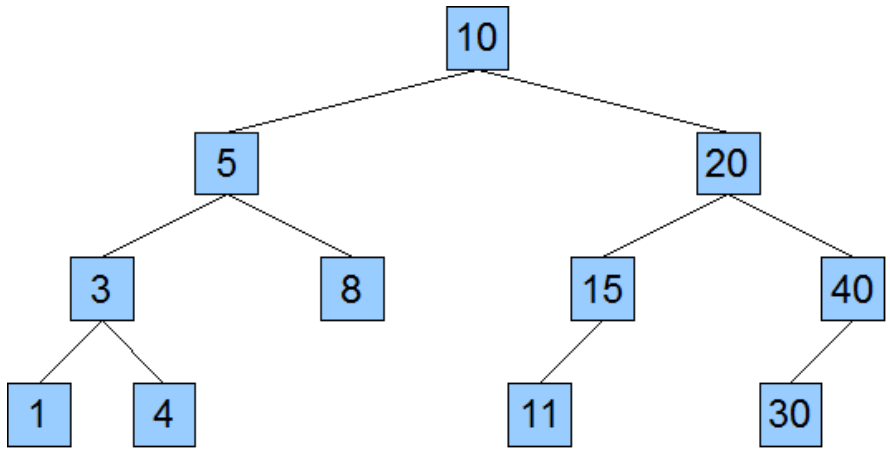
- **Identificador**
  - Nome de variáveis, funções, rótulos e vários outros objetos definidos pelo usuário
- **Constantes**
  - Identificadores que não podem ter seus valores alterados durante a execução do programa

Conceitos básicos VI

- **Estruturas de dados**
  - Tratam da **forma** de se ter **acesso** a **conjuntos de dados** agrupados segundo o algoritmo adotado para a solução de um problema (organização dos dados)

    Encontrar um elemento dentro do conjunto





Conceitos básicos VII

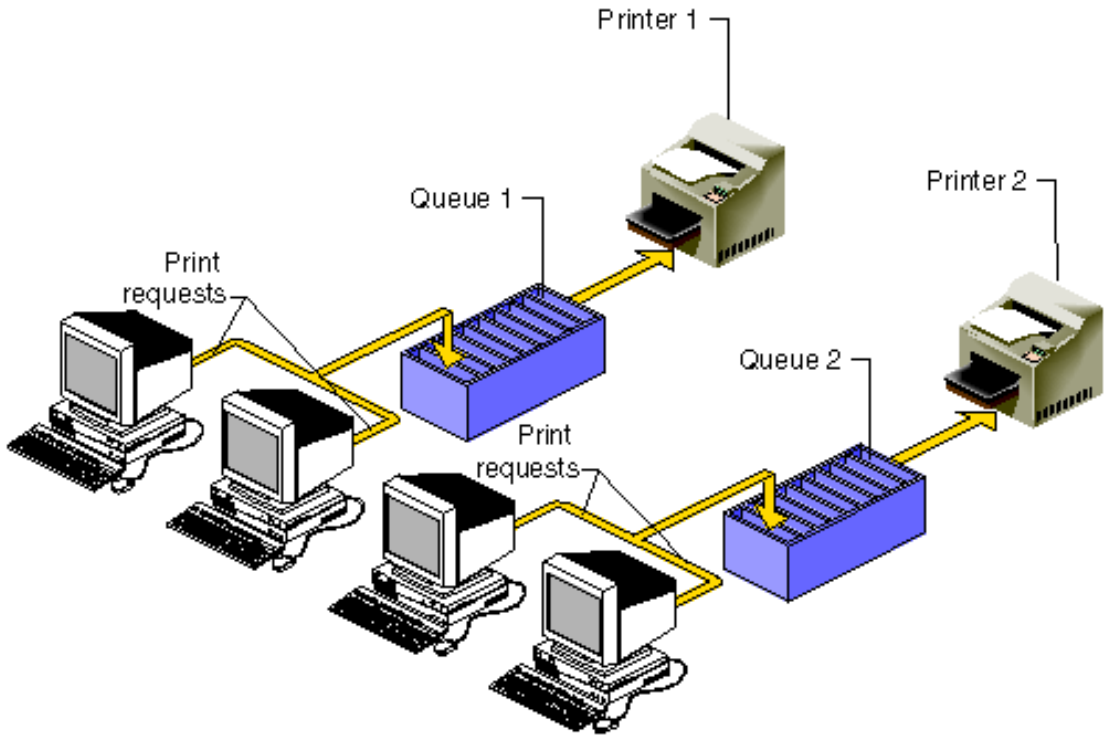
- Objetivo do seu estudo:
  - Teórico (identificar e desenvolver modelos, determinando que classes de problemas podem ser resolvidos com o uso deles)

Conceitos básicos VIII

Como controlar a ordem de impressão de documentos?

Conceitos básicos IX

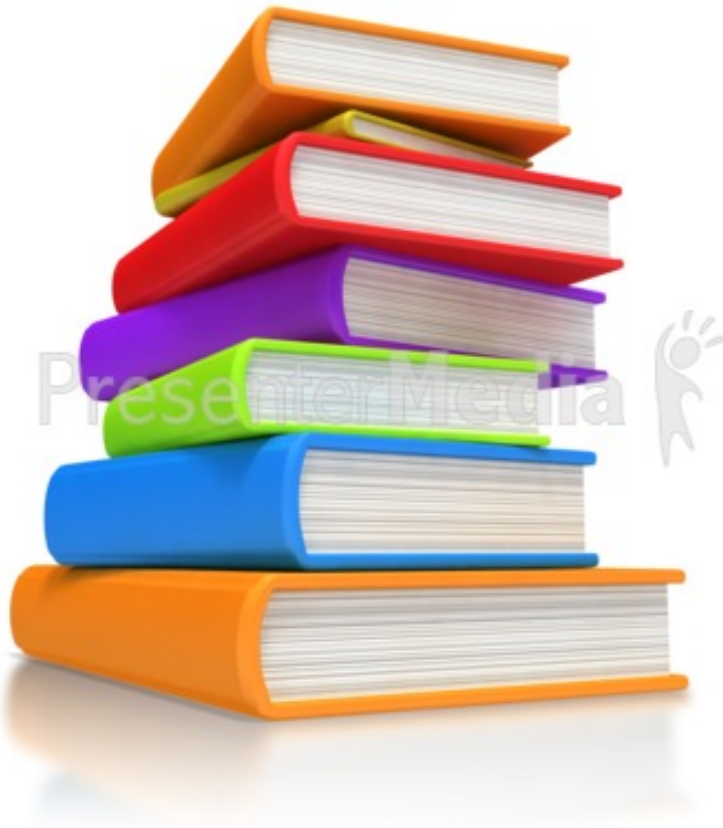
Conceitos básicos X



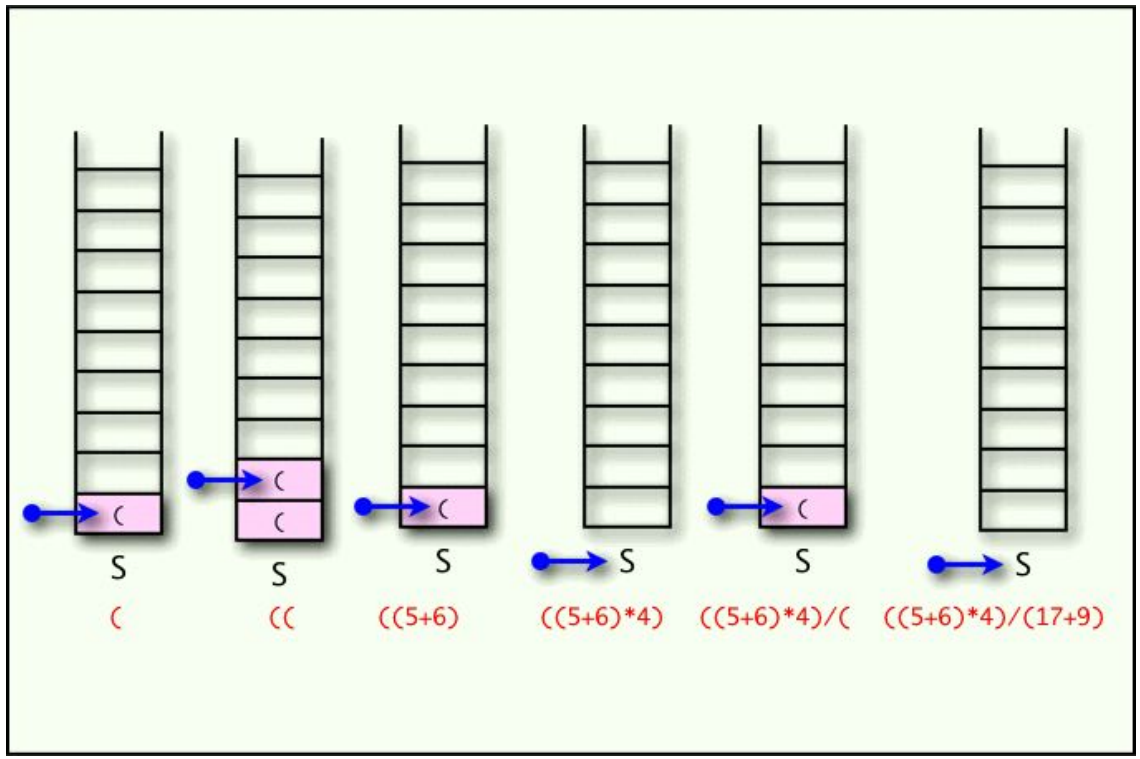
Conceitos básicos XI

Parênteses balanceados dentro de uma equação

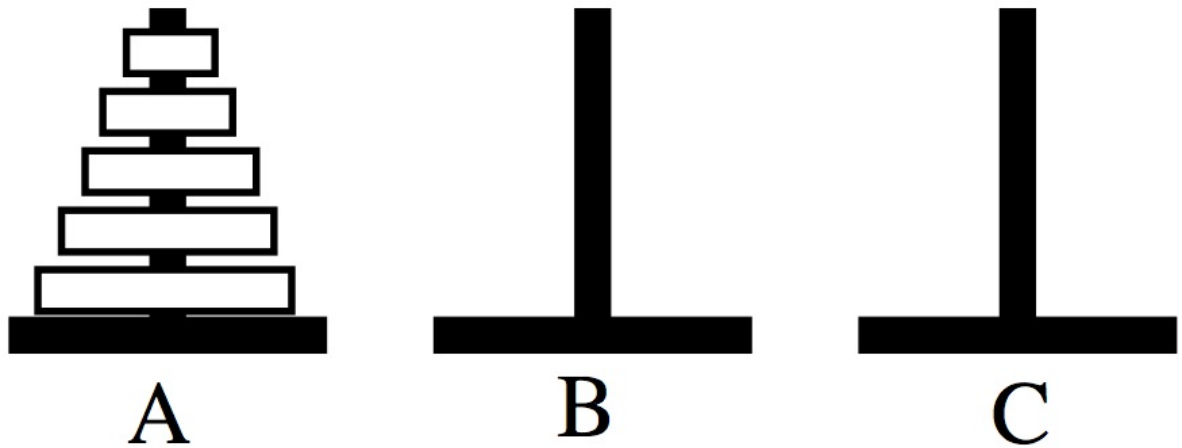
Conceitos básicos XII



Conceitos básicos XIII



Conceitos básicos XIV



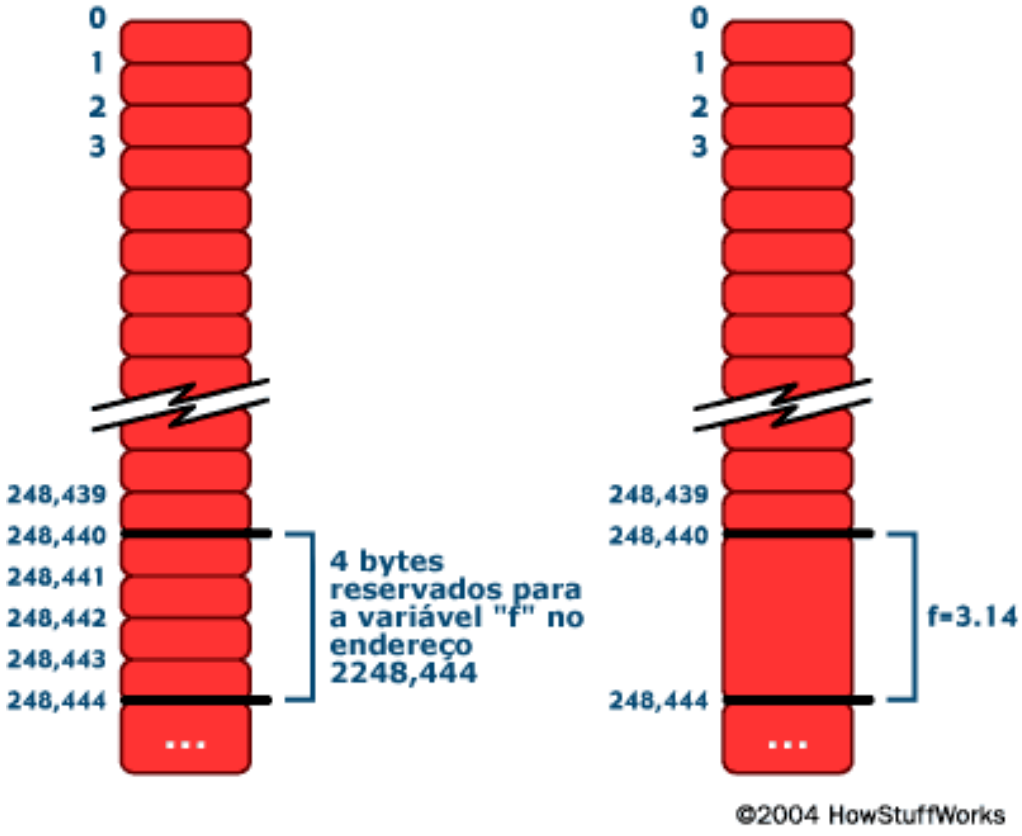
Torres de Hanoi

Conceitos básicos XV

- Prático (criar representações concretas dos objetos e desenvolver rotinas capazes de atuar sobre estas representações)

Conceitos básicos XVI

- **Tipos de dados**
  - Primitivos (inteiro, real, char, ...)
  - Estruturados (vetores, registros, ...)



Conceitos básicos XVII

- **Alocação estática** $\times$ **Alocação dinâmica**
  - Alocação estática
    - Reserva de espaço de memoria antes da execução
    - Variáveis locais e globais armazenadas de forma FIXA
    - Necessidade de previsão de tamanho do espaço (ex. vetor)
  - Alocação dinâmica
    - Reserva de espaço de memoria em tempo de execução
    - Necessidade de funções para alocação
    - Uso de ponteiro para a área reservada e crescimento dinâmico

Conceitos básicos XVIII

- **Algoritmo**
  - Um algoritmo é uma **seqüência de instruções** bem definidas para a **solução de um problema** em um **tempo finito**
  - Um algoritmo **não é a solução** de um problema, **é o caminho** para a solução

Conceitos básicos XIX

- Todo algoritmo deve satisfazer:
  - Entrada: zero ou mais valores de entrada
  - Saída: pelo menos um valor deve ser produzido
  - Claridade: todo instrução deve ser clara e não ambígua
  - Término: o algoritmo deve terminar depois de um número finito de passos
  - Efetividade: toda instrução deve ser factível

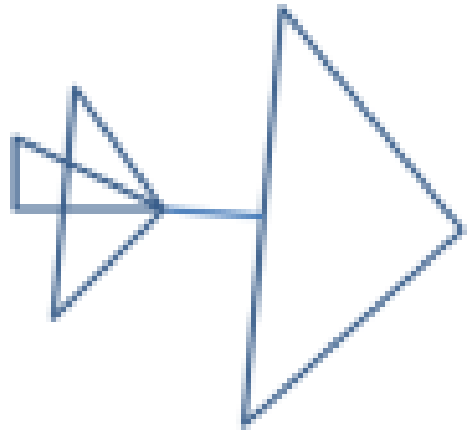
Conceitos básicos XX

- **Programação:** ato de escrever um algoritmo em alguma linguagem de programação.
  - Programas são **formulações concretas** de **algoritmos abstratos**, baseados em **representações e estruturas** específicas de dados

Conceitos básicos XXI

- **Abstração:** está associada à **remoção dos detalhes** de algo ou alguma coisa, com o intuito de **concentrar foco** em suas **características mais importantes.**
  - Os **dados** ou problemas a serem **processados** pelo computador representam uma **abstração de parte da realidade**.
  - A abstração sugere a distinção que deve ser feita entre **o que** o programa faz e **como** ele é implementado.

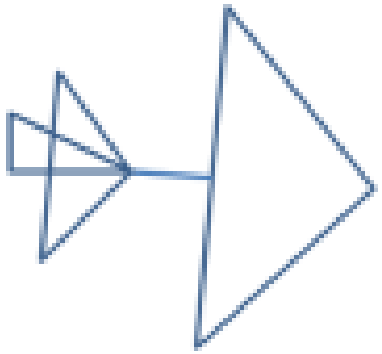
Conceitos básicos XXII



Concreto: todos os detalhes da realidade

Abstrato: só características importantes

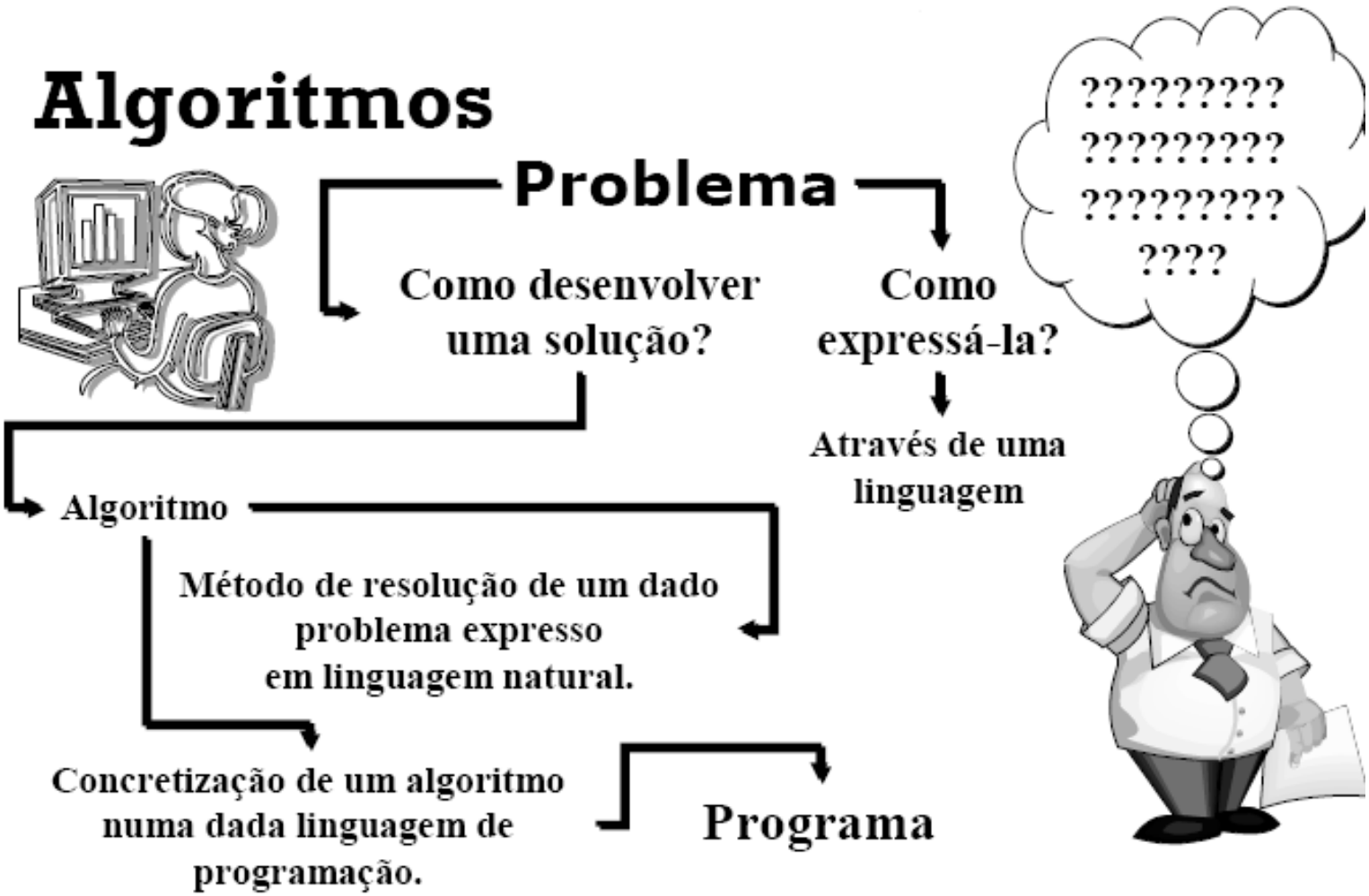
Conceitos básicos XXIII



Quanto mais **abstrato** mais geral.
A figura ao lado poderia representar qualquer avião.

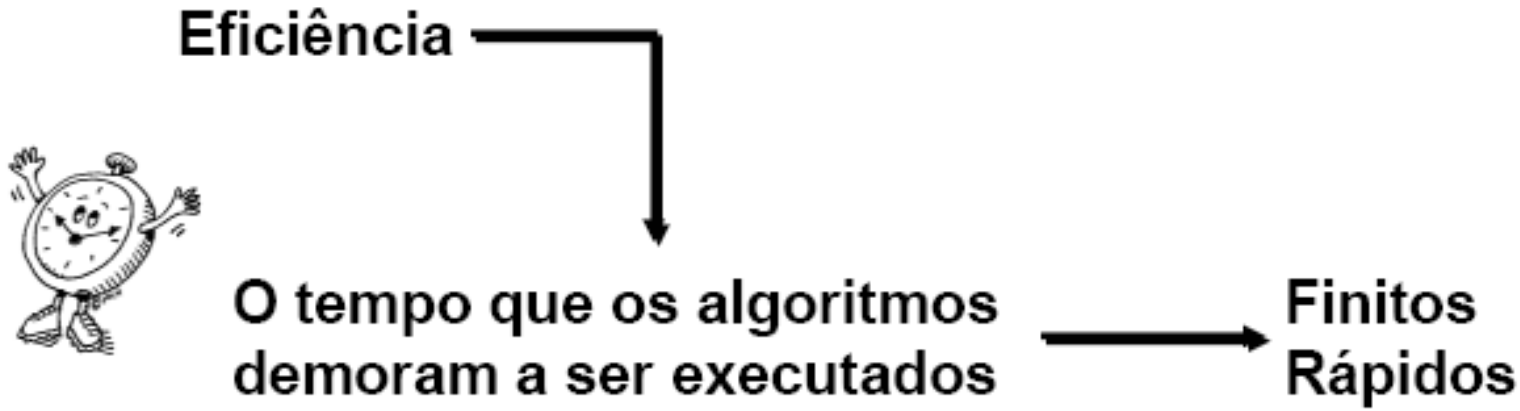
Quanto mais **concreto**, mais específico
O avião ao lado é tão específico, com tantos detalhes, que só representa ele mesmo

Linguagens de programação I



Linguagens de programação II

- Características:



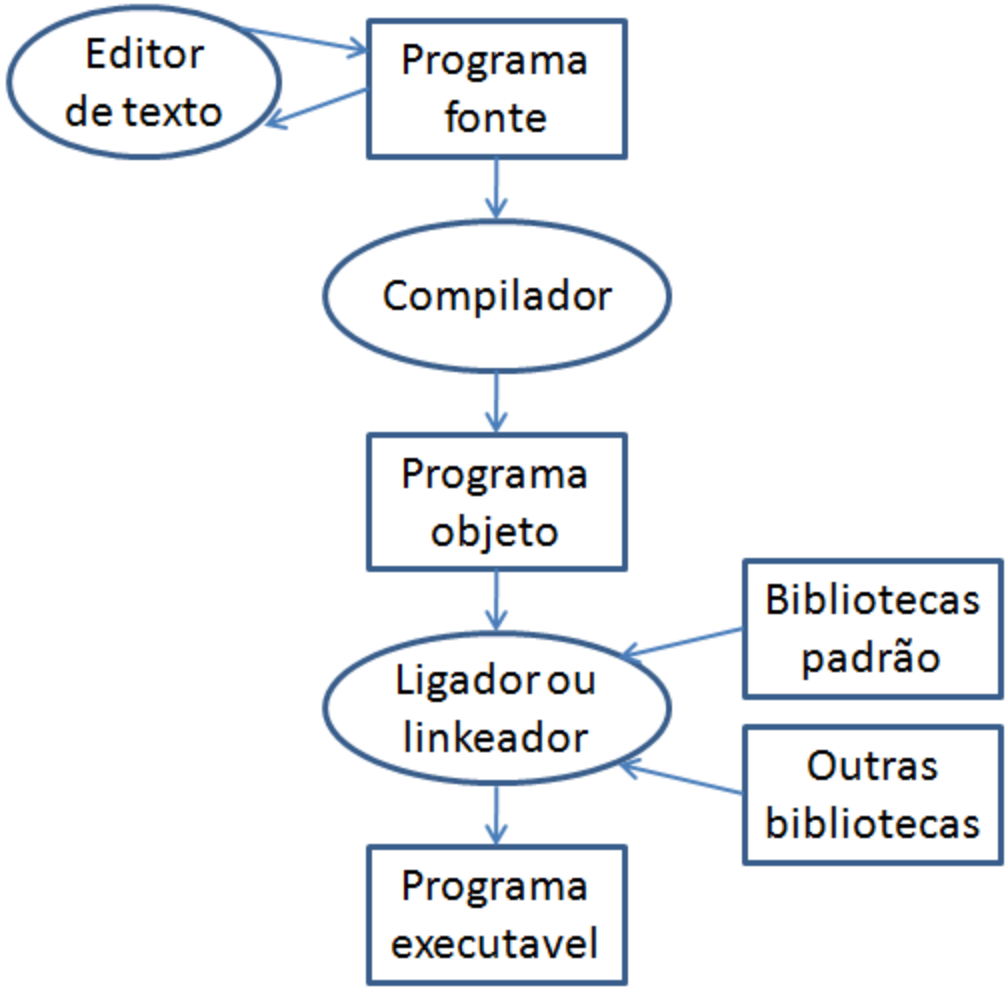
Linguagens de programação III

- Etapas do desenvolvimento de um programa:
  - Análise
    - Estudar o enunciado do problema;
    - Definir os dados de entrada, o processamento e os dados de saída.
  - Algoritmo
    - Descrever o problema com suas soluções.
  - Codificação
    - Transformar o algoritmo numa linguagem de programação escolhida.

Processo de programação I

- Inicia com a edição de um **programa-fonte** e termina com a geração de um **programa-executável**.
- Passos:
  - O programa-fonte é criado em um editor de textos.
  - O compilador analisa o código-fonte e o converte para um código-objeto (versão em linguagem de máquina do programa-fonte).
  - Se o programa contiver chamadas a funções das bibliotecas (função cosseno, por exemplo) o **ligador** (*linker*) junta o programa-objeto com a(s) respectiva(s) biblioteca(s) e gera um código-executável.

Processo de programação II



Processo de programação III

- Principais termos:
  - **Código-fonte**: contem os comando da linguagem de programação
  - **Código-objeto**: criado pela conversão do código-fonte em linguagem de máquina. É gerado pelo compilador.
  - **Ligador ou "linkeador"**: junta o código-objeto com as bibliotecas necessárias para gerar o programa-executável.

Processo de programação IV

- **Programa executável**: código que pode ser executado pelo sistema operacional.
- **Tempo de compilação**: durante o processo de conversão entre código-fonte e código-objeto.
- **Tempo de execução**: após a ativação do programa executável.

Ambiente de Programação C/C++ I

- O código-fonte é editado e armazenado em um arquivo com extensões: `c`, `cpp`
- No código-fonte, todo comando C++ termina com "**;**"
- Em seguida, o programador executa o comando para compilar o código-fonte (arquivo com extensão `obj` ou `o`)

Ambiente de Programação C/C++ II

- Em um sistemas C/C++, o pré-processador é executado automaticamente antes do compilador
  - Obedece comandos especiais chamados de *diretivas do pré-processador*
  - Indicam que manipulações devem ser realizadas no programa antes da compilação (inclusão de outros arquivos no código-fonte)
  - Todas as diretivas começam com #
  - Diretivas do pré-processador não são comandos C/C++, assim elas **não** terminam com "**;**"

Ambiente de Programação C/C++ III

- Diretivas mais utilizadas em C
  - `#include <stdio.h>`, funções de entrada/saída
  - `#include <math.h>`, funções matemáticas
- Diretivas mais utilizadas em C++
  - `#include <iostream>`, funções de entrada/saída
  - `#include <cmath>`, funções matemáticas

Ambiente de Programação C/C++ IV

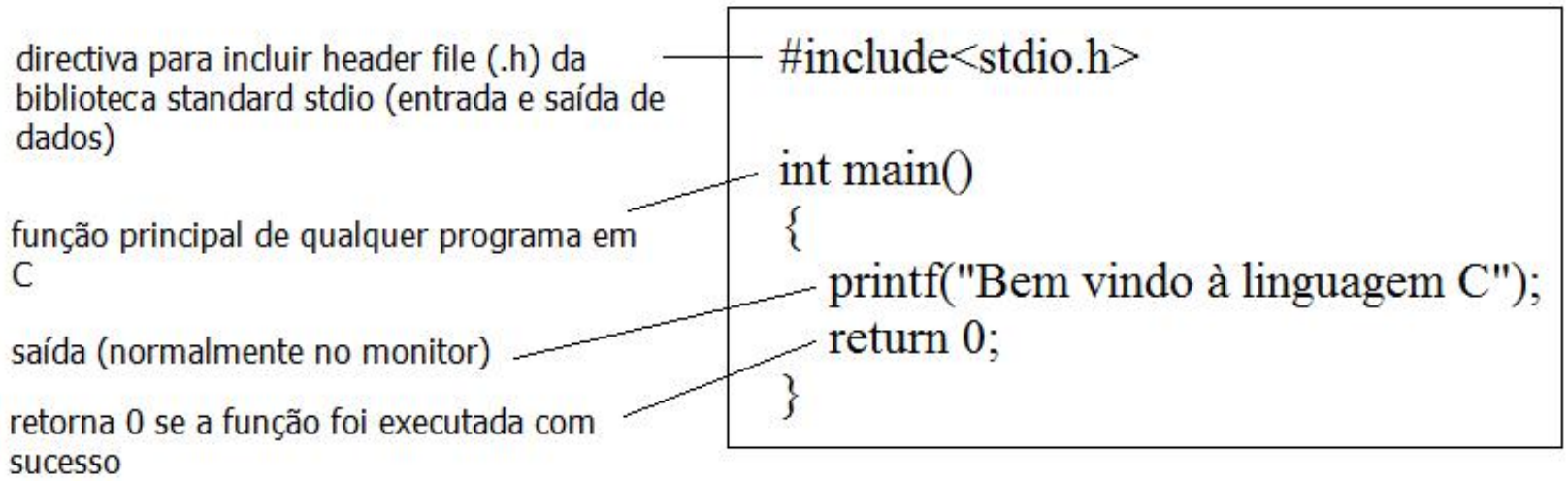
- A próxima fase é chamada de edição de ligações (*link-edition*)
  - Programas C/C++ tipicamente contém chamadas a funções definidas em outros locais, tais como as bibliotecas padrões ou bibliotecas de um projeto particular
  - O código-objeto produzido contém "buracos" devido a essas chamadas
  - O *linker* liga o código-objeto com o código dessas chamadas para produzir o código-executavel.

Exemplo: Programa em C e C++ I

```c
#include <stdio.h>
int main()
{
    printf("Bem vindo à linguagem C! \n");
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
int main()
{
    cout<<"Bem vindo à linguagem C++! \n";
    return 0;
}
```

Exemplo: Programa em C e C++ II



Exemplo: Programa em C e C++ I

- Numérico
  - Inteiro (3, 7, -6, 7829)
  - Real (23.8, 3.789, -78.297, 2.7e10)
- Cadeia de caracteres ou *string* (“abc”, “ana paula”, “3+4=7”)
- Lógico ou booleano (verdadeiro/falso; *true/false*)
- Ponteiro

Exemplo: Programa em C e C++ II

- Exercicios: indique o tipo de cada uma das seguintes constantes
  - 10
  - 10.0
  - -10
  - “10”
  - 6.02e2
  - “fim da questão”
  - true

Operações primitivas I

- Adição e subtração são representadas de forma usual matemática
- Multiplicação
  - Para evitar a possível confusão com a letra **x**, a multiplicação é indicada por um asterisco
- Divisão $\frac{8}{2}$ é representado como $8/2$
- Exponenciação 2^4 é representado por `2^4`

Operações primitivas II

- Exemplos
  - Adição: $2 + 3 + 4$
  - Subtração: $10 - 4 - 1$
  - Multiplicação: $2 * 3 * 4$
  - Divisão: $10{,}0/4$

Operações primitivas III

- Exponenciação: `3^2`
  - Em C/C++ não existe esse operador
  - Ele pode ser substituído pela função embutida `pow(x,y)`
  - C: `#include<math.h>`
  - C++: `#include<cmath>`
  - `3^2` é equivalente a `pow(3,2)`

Operações primitivas IV

- O resultado de uma operação com os dois operandos inteiros é inteiro
- O resultado de uma operação com um operando real e o outro inteiro é real
- Qual é o resultado da expressão?

$$\frac{1}{10} \times 10$$

Operações primitivas V

1/10 * 10
0.1 * 10
0 * 10
0

é truncado para zero pois o resultado da divisão de dois inteiros é outro inteiro

Operações primitivas VI

- Além dos operadores aritméticos convencionais, existem outros operadores ou funções embutidas:
- Resto da divisão: em C/C++ representado por %
- Raiz quadrada:
  - em C/C++ representado por `sqrt(expressão)`
  - Ex: `sqrt(16)` resulta em 4, `sqrt(25+11)` resulta em 6

Expressões relacionais I

- São aquelas que realizam uma comparação entre duas expressões e retornam
  1. Zero (0), se o resultado é falso
  2. Um (1), ou qualquer outro número diferente de zero, se o resultado é verdadeiro

Expressões relacionais II

- Para não esquecer os valores de uma expressão relacional, use:

S1M

NÃ0

Expressões relacionais III

- `<expressao> == <expressao>`: retorna verdadeiro quando as expressão forem iguais
  Ex: $a == b$
- `<expressao> != <expressao>`: retorna verdadeiro quando as expressão forem diferentes
  Ex: $a! = b$

Expressões relacionais IV

- `<expressao> > <expressao>`: retorna verdadeiro quando a expressão da esquerda tiver valor maior que a expressão da direita
  Ex: $a > b$
- `<expressao> < <expressao>`: retorna verdadeiro quando a expressão da esquerda tiver valor menor que a expressão da direita
  Ex: $a < b$

Expressões lógicas I

- São aquelas que realizam uma operação lógica (ou, e, não, etc, . . .) e retorna verdadeiro ou falso
- `<expressao> && <expressao>`: retorna verdadeiro quando ambas expressões são verdadeiras
  Ex. `a == 0 && b == 0`
- `<expressao>` || `<expressao>`: retorna verdadeiro quando pelo menos uma das expressões é verdadeira
  Ex. `a == 0` || `b == 0`
- `! <expressao>`: retorna verdadeiro quando a expressão é falsa e vice-versa.

FIM