---------- AULA 0 ----------

Tipos de Paradigmas

Imperativo

- **Linguagem:** C

Funcional

- **Linguagem:** Clojure

Orientado a Objetos

- **Linguagem:** Java

Lógico

- **Linguagem:** Prolog

Seminário:

O que fazer no seminário:

- Cada dupla com uma linguagem;
- Organização e estruturas da linguagem;
- histórico e influências;
- Tutorial de instalação;
- Desenvolver uma aplicação na linguagem.

Avaliação do Seminário:

- Qualidade do estudo;
- Aplicação desenvolvida;
- Qualidade da apresentação;
- Arguição (perguntas do professor).

Linguagens:

- Python
- C++
- C#
- Javascript
- PHP
- Grovy
- Scala
- Haskell
- Go
- Ruby
- Shell Script
- Lua
- Perl
- Rist
- Dart
- Julia
- Typescript
- R
- SQL
- Elixir
- Common Lisp
- Schema
- Visual Basic
- Swift
- PL/SQL
- Logo
- Fennel

---------- AULA 1 ----------

Linguagem de Programação: Facilita a expressão e a comunicação de ideias entre as pessoas, assim como as linguagens naturais.

- Diferenças importantes:

a) Permite a comunicação de ideias entre pessoas e computadores;

b) Uma Linguagem de Programação possui um domínio de expressão mais restrito;

c) Linguagens de Programação facilita a expressão de ideias computacionais.

Princípios de Linguagens de Programação:

- **Sintaxe:** Descreve o que constitui um programa estruturalmente correto.

Qual é o conjunto básico de palavras e símbolos para escrever um programa estruturalmente correto?

Como um compilador analisa a sintaxe de um programa?

- **Nomes e Tipos:** Uma Linguagem de Programação deve possuir um conjunto de regras para nomear entidades (variáveis, funções, classes, parâmetros, etc).

Propriedades dos nomes: Escopo, visibilidade, ligação.

Tipos: Denotam os tipos de valores que um programa pode manipular:

simples (caracter, inteiro, real, etc);

estruturado (arrays, etc);

complexo (classes, funções).

- **Semântica:** Significado do programa, ou seja, o efeito de cada comando sobre os valores das variáveis.

As funções representam elementos chave da abstração procedural.

* Abstração Procedural é entender o que uma função faz, e não como ela faz isso.

- **Paradigma de Programação:** Modelo ou padrão de resolução de problemas, São basicamente os “tipos” de programação:

Imperativo: Diz **como** as coisas serão feitas;

Orientado a Objeto: Coleção de objetos que interagem entre si, passando mensagens e transformando seus estados.

Funcional: Coleção de funções matemáticas que interagem entre si por meio de composição funcional, condições e recursão.

Lógico: Permite modelar um problema declarando qual resultado o programa deve obter em vez de como ele deve ser obtido (Pode ser visto como uma coleção de regras ou restrições do problema).

---------- AULA 2 ----------

Linguagens tem Características e Critérios:

Critérios:

- Legibilidade
- Facilidade de Escrita
- Confiabilidade
- Custo

Características:

- Simplicidade
- Ortogonalidade
- Tipo de dados
- Abstração
- Expressividade

Restrições de Projeto:

- Arquitetura
- Conf. Técnica
- Padrões
- Sist. Legados

***** Relação Custo-Benefício

Métodos de implementação

- **Compilação:** programas são traduzidos em linguagem de máquina, inclui sistema JIT.
- **Interpretação Pura:** Programas são interpretados por outros programas (interpretadores).
- **Híbrido:** Compromisso entre compiladores e interpretadores puros.

AULA DIA 09/08

- Toda Linguagem de Programação precisa de uma gramática

- Gramática é um conjunto de regras

BNF:

FUNDAMENTOS DA BNF:

- LHS: Left-Hand Side é a abstração definida
- RHS: Right-Hand Side é a definição da abstração. Consiste em um misto de tokens, lexemas e outras abstrações.
- Regra ou produção: é a representação completa (LHS e RHS)
- Símbolos não-terminais: abstração
- Símbolos terminais: lexemas e tokens
- Uma descrição BNF (gramática) é uma coleção de regras.
- Um símbolo inicial (start symbol) é um elemento especial de símbolos não-terminais de uma gramática.

---------- AULA 3 ----------

GRAMÁTICA AMBÍGUA:

Uma gramática é ambígua quando ela gera uma sentença de duas ou mais árvores sintáticas (parse trees) diferentes.

Pode causar um problema para o compilador, pois ele pode ficar em dúvida de qual árvore usar. Então, é recomendado que uma gramática não seja ambígua.

---------- AULA 4 ----------

ANÁLISE LÉXICA E ANÁLISE SINTÁTICA

- Análise de sintaxe c/ análise léxica (baseada em autômatos finitos) =>Separa as entidades / unidades do código
- Análise baseada na gramática BNF.

---------- AULA 5 ----------

ANÁLISE LÉXICA

- Identifica o que as coisas são;
- Exemplo: “A é um indentificador”, “+ é um operador”, etc.

- É um identificador de padrões (pattern matcher) para caracteres de uma string;
- identifica substrings do código-fonte (lexemas), e associa os lexemas aos tokens;

Abordagens para construir um analisador léxico:

- Tabela gerada a partir de uma descrição formal;
- Diagrama de estados que descreve tokens;
- Diagrama de estados que descreve tokens e uma tabela de estados;

ANÁLISE SINTÁTICA

- Vai tentar montar a árvore sintática (ou seja, vai analisar se o programa está feito de forma que faça sentido);

PORQUE SEPARAMOS O ANALISADOR LÉXICO DO SINTÁTICO

- Torna mais simples as análises;
- Permite otimizar o analisador léxico;
- Melhora a portabilidade.

LINGUAGENS IMPERATIVAS:

- Linguagens imperativas são abstrações da arquitetura de von Neumann;
- Variável é abstração de uma célula da memória;
- Variáveis são caracterizadas por uma 6 atributos:
    - Nome
    - Endereço
    - Tipo
    - Valor
    - Tempo de vida (vinculação)
    - Escopo

---------- AULA 6 ----------

Paradigma Imperativa

→ como realizar algo

VINCULAÇÃO

ESTÁTICA: ocorre em tempo de compilação, antes da execução, permanecendo inalterada.

DINÂMICA: Ocorre em tempo de execução, ou seja, pode ser alterada durante a execução.

- Vinculação (binding) de atributos a variáveis

  → Associa sêxtupla a variável

- Vinculação de Tipos

  → Aspectos Importantes:

  - Como o tipo é especificado?
  - Quando a vinculação ocorre?

  → Declaração explícita (vinculação estática): lista nome e especifica o tipo

  → Declaração implícita (vinculação estática): usa convenções ao invés de instruções

Vinculação dinâmica de tipo

→ O tipo é determinado durante a execução do programa;

→ vínculo é temporário

→ vantagem: flexibilidade na programação

→ desvantagem: diminui a capacidade de encontrar erros

⇒ Vinculação de armazenamento e tempo de vida

Alocação: Vincula área de memória a uma variável (reserva);

Desalocação: Desvincula a área de memória (libera);

Tempo de vida: tempo entre a alocação e desalocação;

- 4 Categorias:
  - Variáveis estáticas
    - Alocação de memória antes da execução do programa;
    - Variáveis globais: acessíveis em todo o programa;
    - Variáveis locais estáticas: declaradas dentro de um subprograma, mas deve reter valores entre execuções separadas do subprograma (variáveis sensíveis a história);
    - Vantagem: Endereçamento é direto, não exige custo adicional p/ alocação/desalocação;
    - Desvantagem: Pouca flexibilidade (recursão).
  - Variáveis dinâmicas de pilha
    - A alocação de memória da variável é feita quando a declaração dela é elaborada mas o tipo é amarrado estaticamente;
  - Variáveis dinâmicas do heap (explícitas)
  - Variáveis dinâmicas do geap (implícitas)

---------- AULA 7 ----------

Last
In
First
Out

Escopo

- Vinculação de nome
- Determina a visibilidade de uma variável
- Visibilidade (Capacidade da variável ser referenciada por uma instrução

Escopo de uma variável: Consiste no conjunto de instruções nas quais a variável é visível.

→ Escopo estático: compilador verifica

→ ocorre em tempo de compilação

→ escopo léxico

→ Escopo dinâmico: definido em tempo de execução

→ depende da sequência de execução das instruções

Escopo Estático Exemplo:

```
void f1(){
    int a; (ESCOPO DE "a" COMEÇA AQUI)
    a = a + 1;

    int b; (ESCOPO DE "b" COMEÇA AQUI)
    b = b + 1;
}
```

Variável Local: Uma variável é local a uma unidade de código se for declarada nele.

Variável Não-Local: é uma variável declarada fora da unidade de código mas é visível dentro dela.

- **OBS:** variável global é um caso particular de variável não-local

---------- AULA 8 ----------

A) Bloco Monolítico = Todas variáveis globais
B) Blocos Não-aninhados (disjuntos)
C) Blocos Aninhados

Escopo Dinâmico: A ordem das funções podem alterar a visibilidade das variáveis.

Escopo Estático: “Abriu e fechou Chaves, isso vira um bloco e o tempo de vida da variável é isso”

Escopo Dinâmico: O tempo de vida da variável é o tempo de execução da função, não importa se lá dentro ele chama outra função, se a função não acabou, a variável continua ‘viva’.

---------- AULA 9 ----------

SEMINÁRIO

ESTUDO DA LINGUAGEM (Pronto 17/10):

- Histórico;
- Instalação (**Windows**, Linux);
    - IDE opcional;
- Tipos de Dados / Escopo;
- Estruturas (Dicionário, Lista, etc);
- Estruturas de Controle (If, For, Switch, While, etc);
    - Tem algo diferente?
- Recursos da linguagem (exemplo: Corotinas do C++; Pattern Matching…)**;**
- **Não é pra falar de Biblioteca;**
- Importante fazer exercícios simples como:
    - Helloworld;
    - Média;
    - if;
    - for;
- Colocar curiosidades na apresentação;
    - Onde foi utilizada, etc
- Ideias:
    -

APLICAÇÃO (A definir entre: 01 a 15 de Outubro):

---------- AULA 10 ----------

Subprogramas:

- Essência da programação estruturada/imp.
- Abstração do processo

Abstração: Focar nas informações relevantes dentro de um contexto

“ Basicamente, subprograma é pegar uma função e focar em como construir ESSA função, e não o programa inteiro, como por exemplo, no caso do printf, focar em como imprimir algo, e não o que vai imprimir.

- O reúso de subprogramas resulta em economia de espaço de memória e tempo de codificação, aumenta a legibilidade, enfatiza a estrutura lógica e oculta detalhes de baixo nível.

- **TIPOS DE SUBPROGRAMAS:**
  - **Função:** Retorna Valor
  - **Procedimento:** Não Retorna Valor.

- Se uma função é um modelo fiel das funções matemáticas, ela não produzirá efeitos colaterais
  - Efeitos Colaterais: Não modifica seus parâmetros nem quaisquer variáveis definidas fora dela.
- “É melhor ter 100 funções operando sobre uma estrutura de dados do que 10 funções operando sobre 10 estruturas de dados”, Alan Perlis.
- Programação em blocos monolíticos
  - inviabiliza grandes sistemas
- Dividir para conquistar
  - Dividir um problema em problemas menores
  - Permite a reutilização
  - Facilita o entendimento do programa
  - Segmenta o programa
- Módulo
  - Unidade de Código que pode ser compilado separadamente
  - Deve possuir um objetivo único
  - Deve ser reutilizável
  - Deve interagir com outros módulos (Interface bem definida}

- Abstração
    - Habilidade de Selecionar e representar aspectos essenciais de um contexto específico;
    - Abstração pode ser em diferentes níveis:
        - Abstração Dados
        - Abstração Processos (subprograma)
    - {O que o trecho de código faz?
    - {Como o trecho ou código faz isso.

- Subprogramas
    - Subprograma é uma abstração de processo
    - Permite segmentar o programa em vários blocos lógicos
    - Deve possuir um propósito único e claro
    - Legibilidade, reutilização, manutenção

- Algumas linguagens diferenciam as funções:
    - função: retorna um valor
    - procedimento: não retorna valor
        - Subrotina
    - Linguagem C não faz distinção (Não tem palavra reservada)
        - Procedimento: uma função que retorna void.
    - Em C++ e Java, um método é uma função declarada dentro de uma classe.

- Parâmetros:
    - Meio pelo qual um subprograma pode acessar dados.
        - Ex: float media **(float a, float b)**;
    - Ausência de parâmetros reduz:
        - Legibilidade
            - Ex: float media();
        - Confiabilidade

- Parâmetros: Reais, formais, argumento
    - Parâmetro formal: Identificador listado no cabeçalho do subprograma é usado no corpo dele.
        - Ex: float media(float a, float b)**{**
        - **return (a+b)/2.0;**
        - **}**

- Parâmetro Real: Identificador ou expressões usados na chamada do subprograma
    - Ex: int main() {
    - float x = 10;
    - float y = 15;
    - float m = media (**x**, **y**);
    - // resto do programa
    - return ;
    - }
- Argumento: é o valor passado do parâmetro real p/ o parâmetro formal;
- Correspondência entre parâmetro reais e formais
    - Parâmetros posicionais
    - Parâmetros chave-valor
    - Mix dos dois

- Parâmetro posicional:
    - Os parâmetro reais são relacionados c/ os parâmetros funcionais pela ordem;

- Parâmetro chave-valor:
    - Informa qual parâmetro funcional irá corresponder ao parâmetro real.
        - Ex: media (float a, float b = 1)
        - na chamada:
        - media (x); // assume b=1
        - media (x,y); // b assume y
        - Ex 2:
        - na chamada:
        - media (a=y, b=x)
        - media (b=x, a=y)

- Recursão:
- “Para entender recursão, primeiro é preciso entender recursão!”

⇒ ***Problema c/ estrutura recursiva***

Cada instância do problema contém uma instância menor do mesmo problema

Algoritmo Recursivo

Se: A instância em questão é pequena, resolva imediatamente

Senão

a) reduza a uma instância menor do mesmo problema
b) Aplique o método a instância
c) Volte a instância original b

---------- AULA 11 ----------

MODELOS DE PASSAGEM DE PARÂMETROS

- Processo no qual os parâmetros formais assumem seus respectivos valores;

- 3 aspectos importantes:
  a) direção da passagem
  b) mecanismo de implementação
  c) momento no qual a passagem é realizada

- Tipos de passagem:
  1) Por cópia-valor: Apenas são copiados para novas variáveis, logo modificar o valor do parâmetro formal não causa impacto nos parâmetros reais.
    - preocupação com o espaço de memória.
    - direção: entrada

  2) Por cópia-resultado: Parâmetros formais são copiados para parâmetros reais imediatamente antes do retorno da chamada do subprograma.
    - direção: saída.

  3) Por cópia-valor-resultado: Usar **1** e **2.**

4) **Por referência:** Unidade chamada passa para a unidade chamada <u>endereço</u> do parâmetro real
    - *<u>Benefício:</u>* Otimiza o uso da memória
    - *<u>Problemas:</u>* Legibilidade / Confiabilidade
    - parâmetro formal é um apelido
    - modelo entrada-saída

5) **Por nome:** Amarração do parâmetro a posição não é feita na hora da chamada, mas a cada vez que ele é usado na unidade chamada.
    - Tardia

Implementação de Subprograma

- Quando o subprograma é chamado:
    - Mecanismo do método de passagem de parâmetros (ex: cópia-valor, referência, nome, etc)
    - Alocar dinamicamente as variáveis locais
    - Salvar o estado da unidade chamadora
    - Acesso as variáveis não-locais
- Quando o subprograma encerra:
    - Cópia dos parâmetros de Saída
    - Desalocar as variáveis locais
    - restabelecer o estado da unidade chamadora
    - restabelecer o acesso às variáveis locais
    - devolver o controle à unidade chamadora.
- O que precisa ser armazenado para se realizar o controle:
    - Informação estado sobre a unidade chamadora
    - parâmetro
    - endereço de retorno
    - Valores de retorno para funções
    - Variáveis temporárias usadas pelo código

      (estes dados vão ficar em uma estrutura chamada “registro de ativação”).

- Instância de Registro de Ativação (IRA)

---------- AULA 12 ----------

Paradigma Orientado a Objeto

- Sistema composto por objetos que interagem entre si.
- Objetos possuem atributos e comportamentos.
- Classe representa um conjunto de objeto,

- Exemplo:
  - Mouse:
    - Atributos: Cor, resolução, tipo do scroll, etc
    - Comportamentos: Mover, clicar, rolar
  - Monitor:
    - Atributos: Cor, formato, material, resolução, etc
    - Comportamentos: ligar, desligar

---------- AULA 12 ----------

Abstração de Dados ⇒ Classe (composta por atributos e métodos)

→ Representa um conjunto de objetos

→ Tipo Abstrato de Dados

* Objeto é uma instância da classe

---------- AULA 13 ----------

Um linguagem de programação funcional é considerada **pura** quando ela não possui efeitos colaterais (quando ocorre uma função modifica um de seus parâmetros ou uma variável global)

Ex:

Possui efeito colateral (está alterando o valor da variável 'a'):

```
int a = 0;

int fn(int x){
    a = 10;
    return (x*2);
}
```

Não possui efeito colateral:

```
int a = 0;

int fn(int x){
    return (x*2);
}
```

---------- AULA 14 ----------

DEFN:

(defn soma [a b]

MAP:

O que faz: Mapeia a função para uma lista de valores, gerando a imagem para esta função.

Sintaxe: (map funcao (lista))

- **Exemplo:**

```
(defn dobro [x]
    (* 2 x)
)

(map dobro '(1 2 3 4))
    //RETORNO (2 4 6 8)
```

- **Exemplo em função anônima:**

```
(map (fn [x] (* 2 x)) '(1 2 3 4))
    //RETORNO: (2 4 6 8)
```

REDUCE:

O que faz: Reduz uma quantidade de elementos para apenas 1 elemento aplicando uma função.

Sintaxe: (reduce funcao (lista))

- **Exemplo:**

```
(reduce + '(1 2 3 4))
    //RETORNO: 10
```