Programação

Flavius Gorgônio
flavius@dct.ufrn.br

Detalhes da disciplina

Ementa:

> Métodos de implementação de linguagens de programação: interpretação, compilação, processo híbrido. Sintaxe e semântica. Tipos de dados primitivos e compostos. Escopo e tempo de vida. Vinculação estática e dinâmica. Modularização. Passagem de parâmetros. Alocação dinâmica de memória. Armazenamento de dados em arquivo. Paradigmas de programação: imperativo, funcional, lógico, orientado a objetos. Concorrência e paralelismo. Estudo comparativo de linguagens.

Carga horária:

> 90 horas, sendo 6 aulas/semana

Horário das aulas:

> 234T34

Objetivos e justificativa

Objetivo da disciplina:

Estudar conceitos básicos de programação de computadores, através da implementação de programas de pequeno e médio porte com a utilização de linguagens de programação de alto nível

Justificativa e contexto:

- Breve introdução a algoritmos e lógica de programação no primeiro semestre
- Diversos detalhes de implementação foram omitidos e muitos conceitos foram simplificados
- Os alunos precisam compreender melhor o processo de desenvolvimento de programas de computador
- Iniciar o uso de ferramentas de desenvolvimento realmente utilizadas no desenvolvimento de programas

Plano de Curso

- Introdução e motivação
- Sintaxe e semântica em linguagens de programação
- Tipos de dados, constantes e variáveis
- Expressões aritméticas e lógicas
- Estruturas de controle de decisão
  - Estrutura if/else
  - Estrutura switch
- Estruturas de controle de repetição
  - Estrutura for
  - Estrutura while
  - Estrutura do/while
- Tipos de dados homogêneos
  - Vetores
  - Matrizes
- Subprogramas
- Passagem de parâmetros
- Variáveis globais e locais
- Tipos de dados heterogêneos
  - Registros
  - Uniões
- Tipos de dados enumerados e conjuntos
- Arquivos e registros
- Recursividade
- Alocação dinâmica de memória
- Programação paralela e concorrente
- Paradigmas de programação
  - Imperativo
  - Orientação a objetos
  - Lógico
  - Funcional

Conhecimento exigido

O aluno deve possuir os seguintes conhecimentos prévios:

- Raciocínio lógico e algorítmico
- Boa capacidade de abstração
- Domínio de pelo menos uma linguagem de programação
- Bom relacionamento interpessoal e capacidade de trabalho em equipe
- Ser crítico, criativo, reflexivo, autônomo e participativo
- Motivação para aprender novos conteúdos e manter-se constantemente atualizado

Competências e habilidades

Ao final da disciplina, espera-se que o aluno tenha adquirido/desenvolvido as seguintes competências e habilidades:

- Conheça os principais tipos de dados e as principais estruturas de controle comuns à maioria das linguagem de programação
- Saiba criar programas de médio porte, usando uma linguagem similar a C/C++, inclusive com o uso de funções e passagem de parâmetros
- Saiba ler e criar arquivos de texto e binários
- Conheça os mecanismos de modularização de programas, favorecendo a reutilização de módulos
- Saiba construir sub-rotinas recursivas
- Possua noções básicas sobre alocação dinâmica de memória e o uso de apontadores
- Conheça os principais paradigmas de programação existentes

Metodologia e avaliação

Metodologia:

Aulas virtuais, através de encontros síncronos e assíncronos; discussões com a turma sobre técnicas de programação, desenvolvimento de um projeto completo durante o semestre letivo a partir da abordagem Aprendizagem Baseada em Projetos (*Project Based Learning* - PBL).

Avaliação:

O processo de avaliação se dará de forma contínua, ao longo da disciplina, através de avaliações individuais, exercícios no SIGAA e acompanhamento na elaboração dos projetos.

Horário de atendimento:

23T12

Agenda

- Apresentação da disciplina
- Razões para estudar conceitos de linguagens de programação
- Domínios de programação
- Critérios de avaliação de linguagens
- Influências no projeto de linguagens
- Categorias de linguagens
- *Trade-offs* no projeto de linguagens
- Métodos de implementação
- Ambientes de programação

Motivação para o estudo

Por que estudar sobre Linguagens de Programação?

- Aumento da capacidade de expressar ideias
- Maior conhecimento para a escolha de linguagens apropriadas
- Capacidade aumentada para aprender novas linguagens
- Entender melhor a importância da implementação
- Aumento da capacidade de projetar novas linguagens
- Avanço global da computação

Domínios de programação

Áreas de aplicação para as linguagens de programação:

- Aplicações científicas
  - Grande número de computações de aritmética de ponto flutuante; uso de matrizes (Fortran, Python, R, Julia)
- Aplicações empresariais
  - Produz relatório, usa números decimais e caracteres (COBOL, Object Pascal, Java, PHP)
- Inteligência artificial
  - Símbolos em vez de números manipulados; uso de listas ligadas (LISP, Prolog)
- Programação de sistemas de software básico
  - Precisa de eficiência por causa do uso contínuo (C, C++)
- Software para a Web
  - Eclética coleção de linguagens: de marcação (HTML, XML, XHTML), de scripting (Perl, PHP), de propósito geral (Java, Python, Ruby)

Critérios de avaliação de LPs

- Legibilidade: facilidade com a qual os programas podem ser lidos e entendidos
- Facilidade de escrita: facilidade com a qual uma linguagem pode ser usada para criar programas para um dado domínio
- Confiabilidade: conformidade com as especificações
- Custo: o custo total definitivo de uma linguagem

Legibilidade

Simplicidade geral

- Uma linguagem com muitas construções básicas é mais difícil de aprender
- O ideal é que se tenha um conjunto controlável de recursos e construções
- Mínima multiplicidade de recursos
- Mínima sobrecarga de operadores

```
// multiplicidade de recursos em C/C++/Java

cont = cont + 1

cont += 1

cont++

++cont
```

```
// sobrecarga de operadores em C/C++

a = b & c

d = &e

f = g && h
```

https://replit.com/@flaviusgorgonio/MultiplicidadeDeRecursosc

https://replit.com/@flaviusgorgonio/SobrecargaDeOperadoresc

Legibilidade

Ortogonalidade

- Um conjunto relativamente pequeno de construções primitivas pode ser combinado a um número relativamente pequeno de formas
- Cada possível combinação é legal

```
// quebra de ortogonalidade em Assembly

A     Reg1, memory_cell
AR    Reg1, Reg2

ADDL  operand_1, operand_2
```

```
// quebra de ortogonalidade em Java

int a, b, c;
float d, e, f;
byte g, h, i;

a = b + c;
d = e + f;

g = h + i;  // erro de compilação
```

https://replit.com/@flaviusgorgonio/QuebraDeOrtogonalidadejava

Legibilidade

Tipos de dados

- Mecanismos adequados para definir tipos de dados

```
// legibilidade em tipos de dados
// C não possui o tipo boolean

timeOut = 0;
while (!timeOut) {
   ...
   if (...) {
      timeOut = 1;
   }
}

// contornando o problema

#define true  1
#define false 0

timeOut = false;
while (!timeOut) {
   ...
   if (...) {
      timeOut = true;
   }
}
```

Legibilidade

Projeto da sintaxe

- Formato dos identificadores
- Palavras especiais e métodos de formar sentenças compostas
- Forma e significado: construções autodescritivas, palavras-chave significativas

```
// identificadores em ANSI BASIC

A = 5
A$ = "Linguagem Basic"
```

```
// finalizadores em C/C++/Java

while (...) {
   for (...) {
      if (...) {
         if (...) {
            ...
         }
         if (...) {
            ...
         }
      }
   }
}
```

Facilidade de escrita

- Simplicidade e ortogonalidade
  - Poucas construções, número pequeno de primitivas e um pequeno conjunto de regras para combiná-las
- Suporte à abstração
  - A habilidade de definir e usar estruturas ou operações complicadas de forma a permitir que muitos dos detalhes sejam ignorados
- Expressividade
  - Um conjunto de formas relativamente convenientes de especificar as operações
  - Força e número de operadores e funções pré-definidas

Confiabilidade

- Verificação de tipos
  - Testes para detectar erros de tipos
- Tratamento de exceções
  - Interceptar erros em tempo de execução e tomar medidas corretivas
- Utilização de apelidos
  - Nomes distintos que podem ser usados para acessar a mesma célula de memória
- Legibilidade e facilidade de escrita
  - Uma linguagem que não oferece maneiras naturais para expressar os algoritmos requeridos irá necessariamente usar abordagens não naturais, reduzindo a confiabilidade

Custos

Principais custos associados à escolha de uma LP:

- Treinar programadores para usar a linguagem
- Escrever programas (proximidade com o propósito da aplicação em particular)
- Compilar programas (exigências de hardware)
- Executar programas
- Sistema de implementação da linguagem (disponibilidade de compiladores e outras ferramentas gratuitas)
- Confiabilidade baixa leva a custos altos (erros de codificação)
- Manter programas (disponibilidade de programadores)

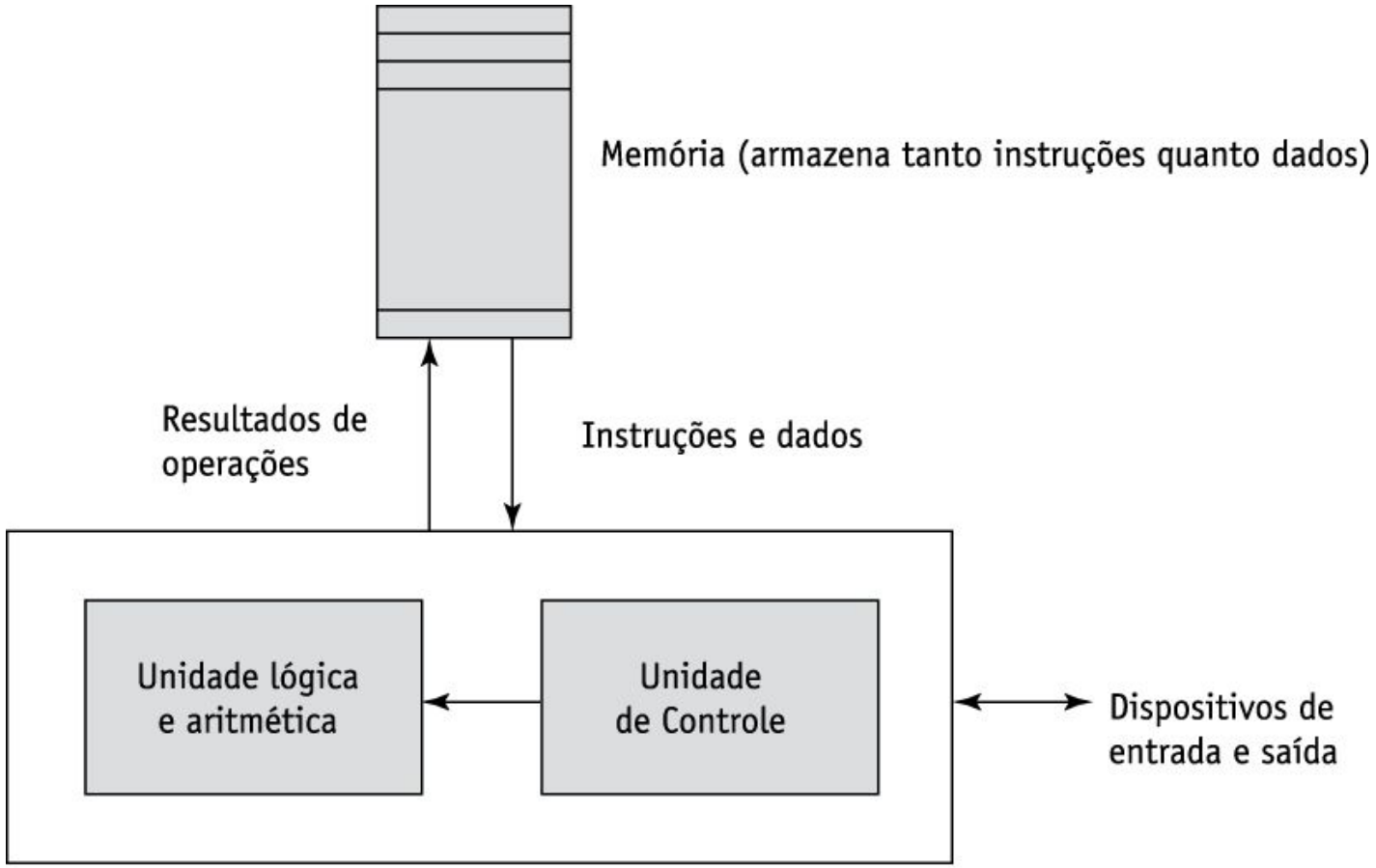
Outros critérios

- Portabilidade
  - A facilidade com a qual os programas podem ser movidos de uma plataforma para outra
- Generalidade
  - A aplicabilidade a uma ampla faixa de aplicações
- Bem definida
  - Em relação à completude e à precisão do documento oficial que define a linguagem

Influências no projeto de LPs

- Arquitetura de computadores
  - Linguagens são projetadas considerando a principal arquitetura de computadores, chamada de arquitetura de von Neumann
- Linguagens imperativas são mais populares por causa dos computadores von Neumann
  - Dados e programas armazenados na memória
  - A memória é separada da CPU
  - Instruções e dados são canalizadas a partir da memória para CPU
  - Base para linguagens imperativas
    - Variáveis modelam as células de memória
    - Sentenças de atribuição são baseadas na operação de envio de dados e instruções
    - Iteração é eficiente

Arquitetura de von Neumann



Ciclo de obtenção e execução

- Em um computador com arquitetura von Neumann

```
inicialize o contador de programa

repita para sempre

   obtenha a instrução apontada pelo contador de programa

   incremente o contador de programa

   decodifique a instrução

   execute a instrução

fim repita
```

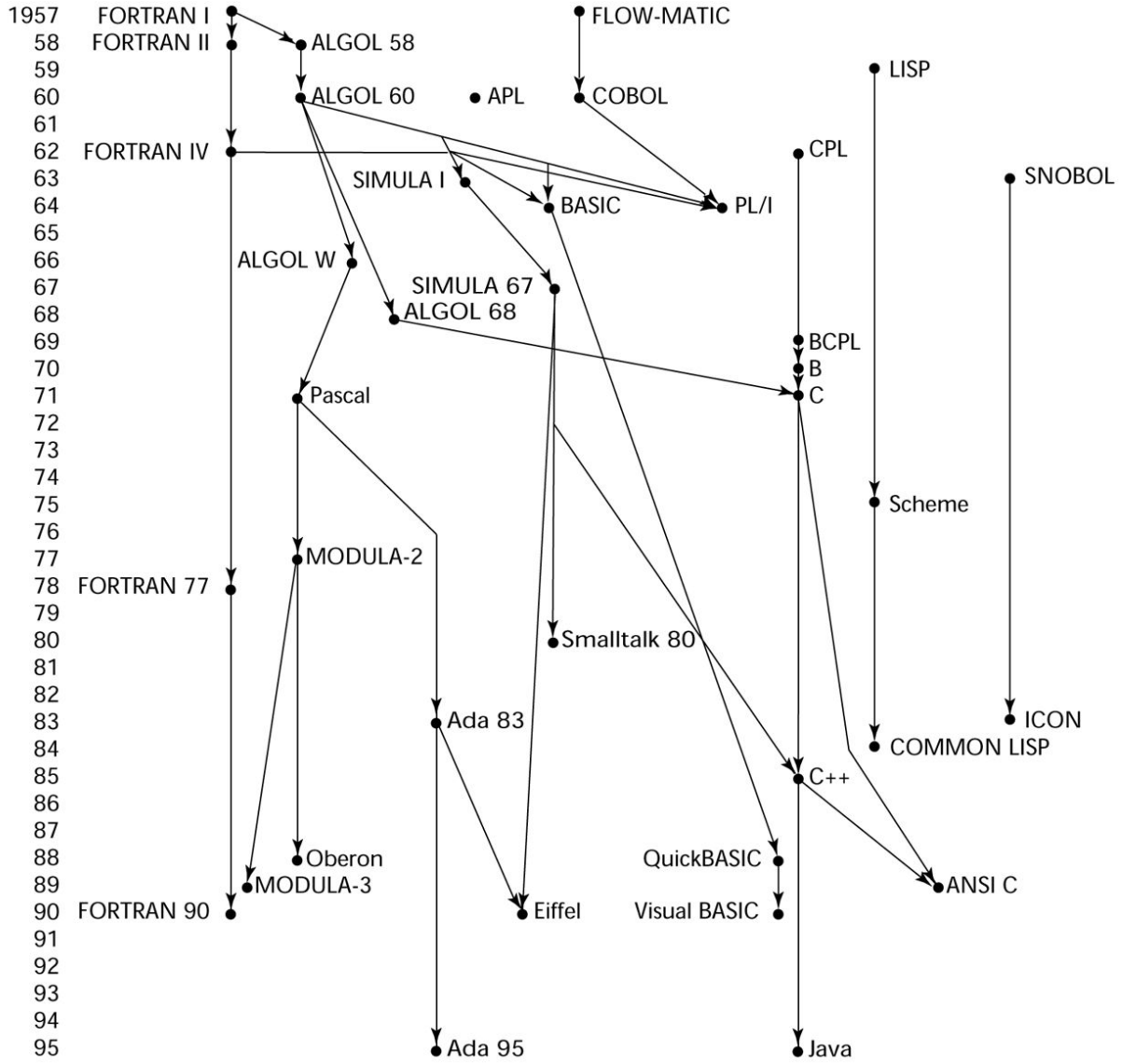
Gargalo de von Neumann

- A velocidade de conexão entre a memória de um computador e seu processador determina a velocidade do computador
- Instruções de programa normalmente podem ser executadas mais rapidamente do que a velocidade de conexão
- Isso resulta em um gargalo, que é conhecido como gargalo de von Neumann
- É o fator limitante primário na velocidade dos computadores

Evolução das LPs

- Anos 1950 e começo dos 1960:
  - Aplicações simples; preocupação com a eficiência da máquina
- Final dos anos 60:
  - Eficiência das pessoas se tornou importante; legibilidade, melhores estruturas de controle
  - Programação estruturada
  - Projeto descendente (top-down) e de refinamento passo a passo
- Final dos anos 70:
  - Da orientação aos procedimentos para uma orientação aos dados
  - Abstração de dados
- Meio dos anos 80:
  - Programação orientada a objetos
  - Abstração de dados + herança + vinculação dinâmica de métodos
  - Novas metodologias de desenvolvimento de software levaram a novos paradigmas de programação e, por extensão, a novas linguagens de programação

Evolução das LPs



Paradigmas de linguagens

- Imperativo
  - Características centrais são variáveis, sentenças de atribuição e de iteração
  - Inclui linguagens que suportam programação orientada a objeto
  - Inclui linguagens de scripting
  - Inclui as linguagens visuais
  - Exemplos: C, Java, Perl, JavaScript, Visual BASIC .NET, C++
- Funcional
  - Principais meios de fazer os cálculos é pela aplicação de funções para determinados parâmetros
  - Exemplos: LISP, Scheme
- Lógico
  - Baseada em regras (regras são especificadas sem uma ordem em particular)
  - Exemplo: Prolog
- De marcação/programação híbrida
  - Linguagens de marcação estendida para suportar alguma programação
  - Exemplos: HTML, JSTL, XSLT

Imperativo (C++)

```
#include <iostream>

using namespace std;

void incluirAluno(void);
void exibirAlunos(void);

void incluirAluno(void) {
 cout << "Módulo Incluir Aluno" << endl << endl;
}
void exibirAlunos(void) {
 cout << "Módulo Exibir Alunos" << endl << endl;
}
```

```
int main(void) {
  char resp = ' ';
  do {
    cout << endl;
    cout << "Gestão da Turma" << endl << endl;
    cout << "1 - Cadastra Aluno" << endl;
    cout << "2 - Exibe Alunos" << endl;
    cout << "0 - Fim do Programa" << endl << endl;
    cout << "Escolha sua opcao: ";
    cin >> resp;
    getchar();
    switch (resp) {
      case '1' : incluirAluno();
                 break;
      case '2' : exibirAlunos();
                 break;
    }
  } while (resp != '0');
  return 0;
}
```

Funcional (Scheme)

```
(define inverteListax
  (lambda (lista1 lista2)
      (if (null? lista1) lista2
      (inverteListax (cdr lista1) (cons (car
lista1) lista2)))))

(define inverteLista
   (lambda (lista)
      (inverteListax lista '())))

(define (perf n)
      (let loop ((i 1)
            (sum 0))
      (cond ((= i n)
            (= sum n))
            (
            (= 0 (modulo n i))
            (loop (+ i 1) (+ sum i)))
            (else
            (loop (+ i 1) sum)))))
```

```
(define aux
   (lambda (x n l)
      (if (= x 0) l
      (if (perf n) (aux (- x 1) (+ n 1) (cons n l))
                  (aux x (+ n 1) l)))))

(define main
   (lambda (x)
      (aux x 1 '())))

(define perfeitos
   (lambda (n)
      (inverteLista (main n))))
```

Lógico (Prolog)

```
split([], K, [], []).

split(XS, K, [], XS) :-
      K < 1.

split([X|XS], K, [X|YS], ZS) :-
      K >= 1,
      P is K -1,
      split(XS, P, YS, ZS).

merge1([], [], []).

merge1(XS, [], XS).

merge1([], YS, YS).

merge1([X|XS], [Y|YS], [X|ZS]) :-
      X =< Y,
      merge1(XS, [Y|YS], ZS).

merge1([X|XS], [Y|YS], [Y|ZS]) :-
      Y < X,
      merge1([X|XS], YS, ZS).
```

```
mergesort([], []).

mergesort([X], [X]).

mergesort([X, Y], [X, Y]) :-
    X =< Y, !.

mergesort([X, Y], [Y, X]) :-
    X > Y, !.

mergesort(XS, ZS) :-
    length(XS, L),
    L > 0,
    K is L / 2,
    split(XS, K, XS1, XS2),
    mergesort(XS1, YS1),
    mergesort(XS2, YS2),
    merge1(YS1, YS2, ZS), !.
```

Marcação/Híbrida (HTML)

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8" />
		<meta name="description" content="a descrição do seu site em no máximo 90 caracteres">
		<meta name="keywords" content="escreva palavras-chaves curtas, máximo 150 caracteres">
		<title>Título do Documento</title>
	</head>
	<body>
		<!-- Aqui fica a página que será visível para todos, onde pode-se inserir
		textos, imagens, links para outras páginas, etc, geralmente usa-se: →
		<div>Tag para criar-se uma 'caixa', um bloco, mais utilizada com "Cascading Style Sheets
		(Folhas de Estilo em Cascata)</div>
		<span>Tag para modificação de uma parte do texto da página</span>
		<img src="endereço_de_uma_imagem.jpg" />
		<a href="http://www.wikipedia.org">Wikipedia, A Enciclopédia Livre</a>
	</body>
</html>
```

Trade-Offs no projeto de LPs

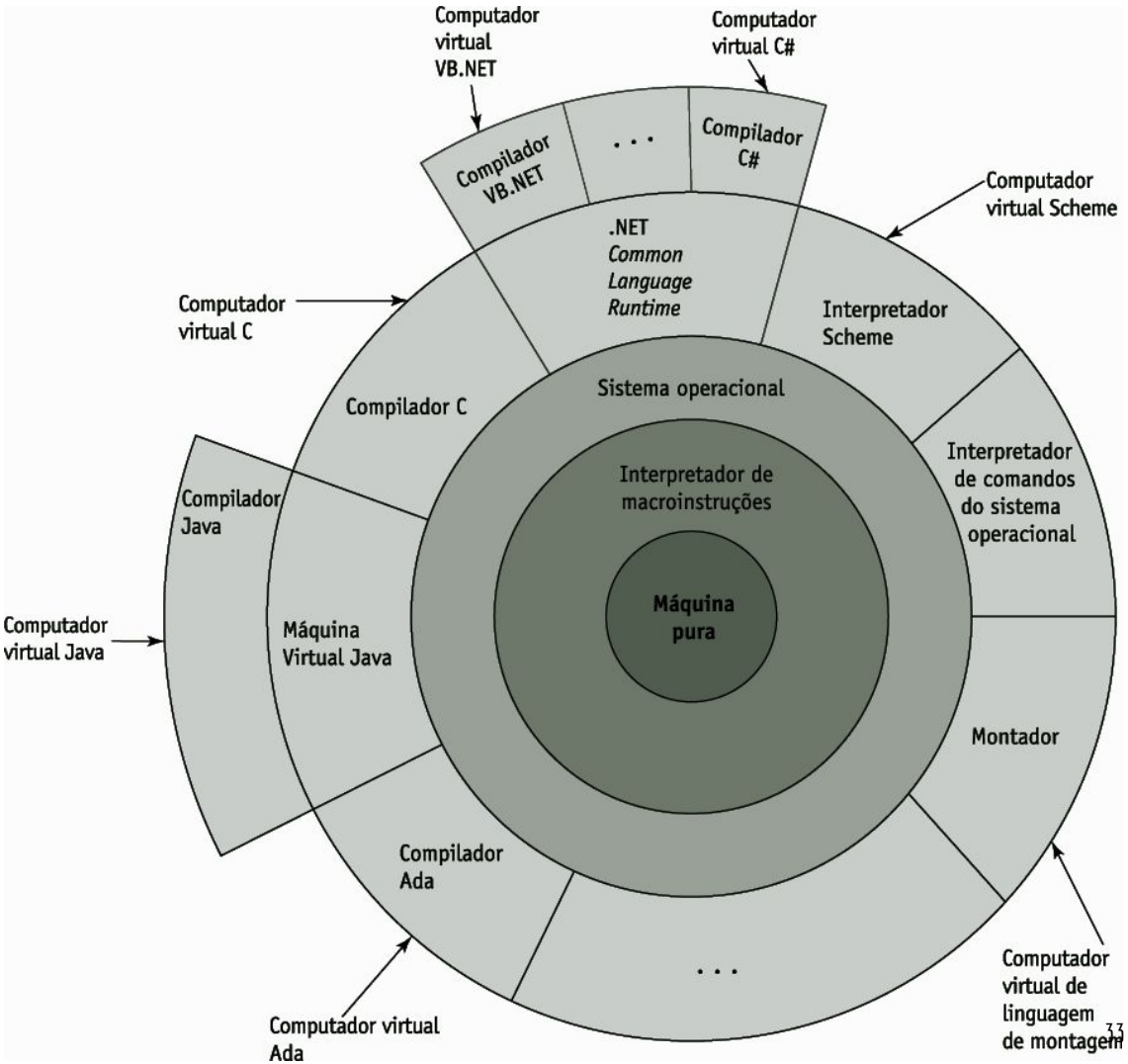
- Confiabilidade × custo de execução
  - Java exige que todas as referências aos elementos de um vetor sejam verificadas para garantir que os índices estão em suas faixas legais
- Legibilidade × facilidade de escrita
  - APL inclui um poderoso conjunto de operadores (e um grande número de novos símbolos), permitindo que computações complexas sejam escritas em um programa compacto, com o custo de baixa legibilidade
- Facilidade de escrita (flexibilidade) × confiabilidade
  - Ponteiros de C++ são poderosos e flexíveis, mas são uma possível fonte de erros de codificação

Métodos de implementação

- Compilação
  - Programas são traduzidos para linguagem de máquina
- Interpretação pura
  - Programas são interpretados por outro programa chamado interpretador
- Sistemas de implementação híbridos
  - Um meio termo entre os compiladores e os interpretadores puros

Computador em camadas

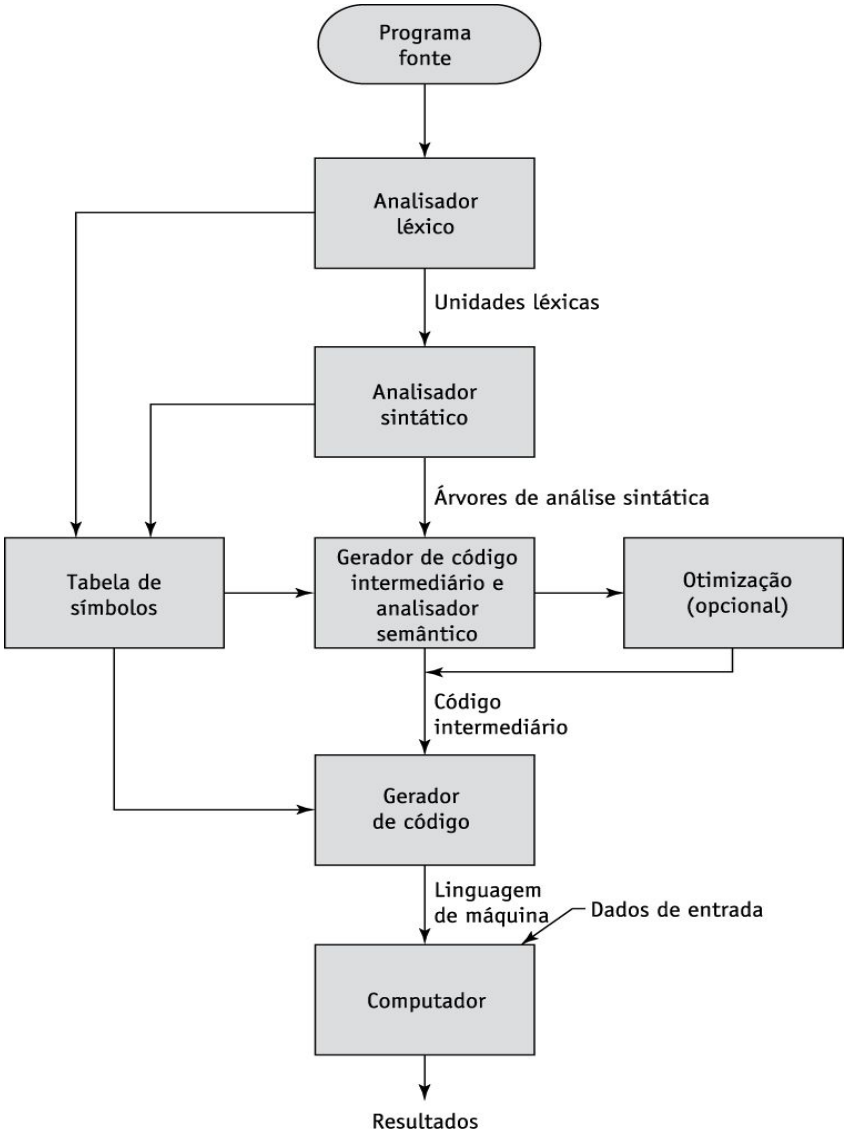
O sistema operacional e as implementações de linguagem são colocados em camadas superiores à interface de linguagem de máquina de um computador



Compilação

- Traduz programas (linguagem fonte) em código de máquina (linguagem de máquina)
- Tradução lenta, execução rápida
- Processo de compilação tem várias fases:
  - análise léxica: agrupa os caracteres do programa fonte em unidades léxicas
  - análise sintática: transforma unidades léxicas em árvores de análise sintática (*parse trees*), que representam a estrutura sintática do programa
  - análise semântica: gera código intermediário
  - geração de código: código de máquina é gerado

Processo de compilação



Terminologias de compilação adicionais

- Módulo de carga (imagem executável): o código de usuário e de sistema juntos
- Ligação e carga: o processo de coletar programas de sistema e ligá-los aos programas de usuário

Um exemplo prático de compilação

1. Abra um editor de texto qualquer
2. Digite o código ao lado e salve-o como:

   media.c

3. Compile o código a partir da linha de comando, usando:

   >> gcc -c media.c

4. Se não houver nenhum erro, gere o código executável usando:

   >> gcc -o media media.o

5. Execute o programa usando:

   >> ./media

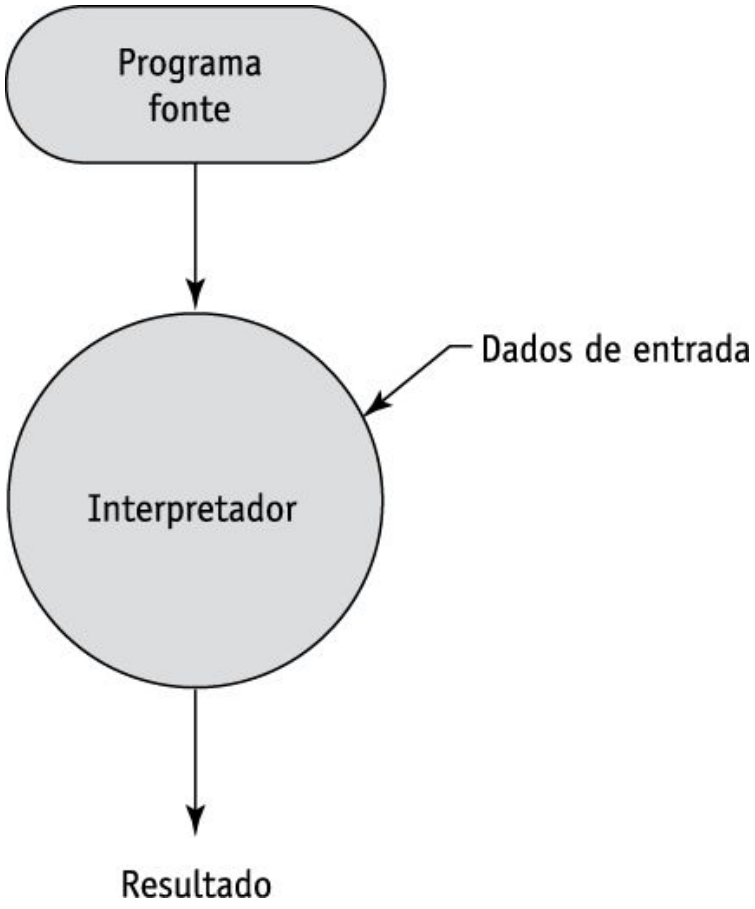
```c
#include <stdio.h>

int main(void) {

    float n1, n2, n3, med;

    printf("Cálculo da média\n");

    printf("Informe a nota 1: ");

    scanf("%f", &n1);

    printf("Informe a nota 2: ");

    scanf("%f", &n2);

    printf("Informe a nota 3: ");

    scanf("%f", &n3);

    med = (n1 + n2 + n3) / 3;

    printf("Média do aluno: %.2f", med);

    return 0;

}
```

https://replit.com/@flaviusgorgonio/Mediac

Interpretação pura

- Não há tradução de código
- Fácil implementação de programas (mensagens de erro em tempo de execução podem referenciar unidades de código fonte)
- Execução mais lenta (tempo de execução de 10 a 100 vezes mais lento do que nos sistemas compilados)
- Geralmente requer mais espaço de memória
- Raramente usada em linguagens de alto nível
- Volta significativa com algumas linguagens de scripting para a web (como JavaScript, PHP e Python)

Processo de interpretação



Um exemplo prático de interpretação pura

1. Abra um editor de texto qualquer
2. Digite o código ao lado e salve-o como:

   media.py

3. Execute o código chamado o interpretador Python a partir da linha de comando:

   >> python media.py

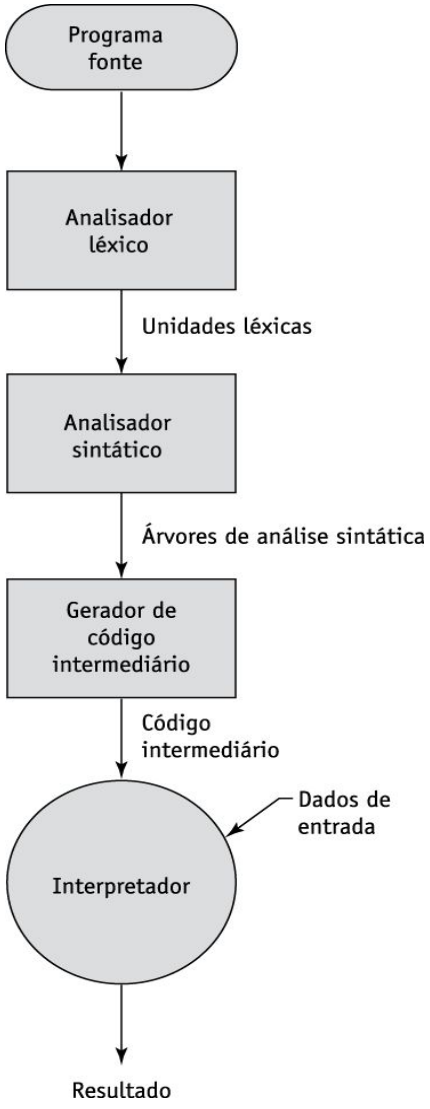
```
print("Cálculo da média")

n1 = float(input("Informe a nota 1: "))

n2 = float(input("Informe a nota 2: "))

n3 = float(input("Informe a nota 3: "))

med = (n1 + n2 + n3) / 3;

print("Média do aluno: %.2f"%med)
```

https://replit.com/@flaviusgorgonio/Mediapy

Implementação híbrida

- Um meio termo entre os compiladores e os interpretadores puros
- Uma linguagem de alto nível é traduzida para uma linguagem intermediária que permite fácil interpretação
- Mais rápido do que interpretação pura
- Exemplos:
  - Programas em Perl eram parcialmente compilados para detectar erros antes da interpretação
  - As primeiras implementações de Java eram todas híbridas; seu formato intermediário, *bytecode*, fornece portabilidade para qualquer máquina que tenha um interpretador de *bytecodes* e um sistema de tempo de execução associado (juntos, são chamados de Máquina Virtual Java)

Implementação híbrida



Um exemplo prático de implementação híbrida

1. Abra um editor de texto qualquer
2. Digite o código ao lado e salve-o como:

   Media.java

3. Compile o código a partir da linha de comando, usando:

   >> javac Media.java

4. Se não houver nenhum erro, o compilador irá gerar um código intermediário chamado:

   >> Media.class

5. Execute o programa usando:

   >> java Media

```
import java.util.Scanner;

public class Media {

    public static void main (String[] args) {

        Scanner ent = new Scanner(System.in);

        float n1, n2, n3, med;

        System.out.println("Cálculo da média");

        System.out.println("Informe a nota 1: ");

        n1 = ent.nextFloat();

        System.out.println("Informe a nota 2: ");

        n2 = ent.nextFloat();

        System.out.println("Informe a nota 3: ");

        n3 = ent.nextFloat();

        med = (n1 + n2 + n3) / 3;

        System.out.println("Média do aluno: " + med);

    }
}
```

https://replit.com/@flaviusgorgonio/Mediajava

Implementação *Just-in-Time*

- Inicialmente traduz os programas para uma linguagem intermediária
- Então, compila os métodos da linguagem intermediária para linguagem de máquina quando esses são chamados
- A versão em código de máquina é mantida para chamadas subsequentes
- Sistemas JIT são agora usados amplamente para programas Java
- As linguagens .NET também são implementadas com um sistema JIT

Simulando erros de compilação em C

1. Modifique o programa anterior de acordo com os trechos destacados ao lado.
2. Repita as etapas de compilação

```c
#include <stdio.h>

int main(void) {

    float n1, n2, n3;

    printf("Cálculo da média\n");

    printf("Informe a nota 1: ");

    scanf("%f", &n1);

    printf("Informe a nota 2: ");

    scanf("%f", &n2);

    printf("Informe a nota 3: ");

    scanf("%f", &n3);

    med = (n1 + n2 + n3) / 3;

    printf("Média do aluno: %.2f", med);

    return 0;

}
```

https://replit.com/@flaviusgorgonio/SimulandoErrosc

Simulando erros de compilação em C

1. Modifique o programa anterior de acordo com os trechos destacados ao lado.
2. Repita as etapas de compilação

```c
#include <stdio.h>

int main(void) {

    float n1, n2, n3, med;

    printf("Cálculo da média\n");

    printf("Informe a nota 1: ");

    scanf("%f", &n1);

    printf("Informe a nota 2: ");

    scanf("%f", &n1);

    printf("Informe a nota 3: ");

    scanf("%f", &n1);

    med = (n1 + n2 + n3) / 3;

    printf("Média do aluno: %.2f", med);

    return 0;

}
```

https://replit.com/@flaviusgorgonio/SimulandoErros2c

Simulando erros de compilação em Java

1. Modifique o programa anterior de acordo com os trechos destacados ao lado.
2. Repita as etapas de compilação

```
import java.util.Scanner;

public class Media {

    public static void main (String[] args) {

        Scanner ent = new Scanner(System.in);

        float n1, n2, n3;

        System.out.println("Cálculo da média");

        System.out.println("Informe a nota 1: ");

        n1 = ent.nextFloat();

        System.out.println("Informe a nota 2: ");

        n1 = ent.nextFloat();

        System.out.println("Informe a nota 3: ");

        n1 = ent.nextFloat();

        med = (n1 + n2 + n3) / 3;

        System.out.println("Média do aluno: " + med);

    }
}
```

https://replit.com/@flaviusgorgonio/SimulandoErrosjava

Simulando erros de interpretação em Python

1. Modifique o programa anterior de acordo com os trechos destacados ao lado.
2. Repita as etapas de interpretação

```
print("Cálculo da média")

n1 = float(input("Informe a nota 1: "))

n1 = float(input("Informe a nota 2: "))

n1 = float(input("Informe a nota 3: "))

med = (n1 + n2 + n3) / 3

print("Média do aluno: %.2f"%med)
```

https://replit.com/@flaviusgorgonio/SimulandoErrospy

Ambientes de programação

Coleção de ferramentas usadas no desenvolvimento de software

- UNIX / Linux
  - Um ambiente de programação mais antigo, porém têm sido mais utilizado por meio de uma interface gráfica com o usuário (GUI)
- Microsoft Visual Studio .NET
  - Grande e complexo
  - Usado para desenvolver software em qualquer uma das cinco linguagens .NET
  - Versão mais enxuta, o Visual Code
- NetBeans / Eclipse
  - Usados primariamente para o desenvolvimento de aplicações Web usando Java, mas também oferece suporte a JavaScript, Ruby e PHP
- Plataformas Web
  - IDE disponíveis on line na web: Repl.it, CodeChef, Ideone, Codeanywhere, etc.

Bibliografia recomendada

CELES FILHO, W. *et al*. Introdução a estruturas de dados. Rio de Janeiro: Elsevier, 2004.

HUNT, A.; THOMAS, D. O programador pragmático: de aprendiz a mestre. Porto Alegre: Bookman, 2010.

KERNIGHAN, B. W.; PIKE, R. The practice of programming. Boston: Addison-Wesley, c1999.

SEBESTA, R. W. Conceitos de linguagens de programação. Porto Alegre: Bookman, 2011.

VAREJÃO, F. M. Linguagens de programação: conceitos e técnicas. Rio de Janeiro: Elsevier, 2004.