Linguagens de Programação

Lucas Ismaily

Universidade Federal do Ceará
Campus de Quixadá

Semestre 2022.2

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

- Aumento da capacidade de expressar idéias;

- Aumento da capacidade de expressar idéias;
  - Capacidade de expressão $\rightarrow$ capacidade intelectual;
  - Como definir o maior inteiro disponível, sabendo que o inteiro ocupa 4 bytes? [$maxInt =\sim (1 \ll 31)$];

- Aumento da capacidade de expressar idéias;
  - Capacidade de expressão $\rightarrow$ capacidade intelectual;
  - Como definir o maior inteiro disponível, sabendo que o inteiro ocupa 4 bytes? [$maxInt =\sim (1 \ll 31)$];
- Maior embasamento na escolha da linguagem apropriada;

- Aumento da capacidade de expressar idéias;
  - Capacidade de expressão $\rightarrow$ capacidade intelectual;
  - Como definir o maior inteiro disponível, sabendo que o inteiro ocupa 4 bytes? [$maxInt =\sim (1 \ll 31)$];
- Maior embasamento na escolha da linguagem apropriada;
  - Python ou C++?;
  - Produtividade ou performance;
  - Eu gosto mais dessa linguagem.

- Capacidade aumentada para aprender novas linguagens;
  - A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original [A. Einstein];

- Capacidade aumentada para aprender novas linguagens;
  - A mente que se abre a uma nova idéia jamais voltará ao seu tamanho original [A. Einstein];
- Entender melhor a importância da implementação;
  - Compreender questões de implementação é importante;
  - Uso de uma linguagem de forma mais inteligente.

- Aumento da capacidade de projetar novas linguagens;
  - Certo estudante queria algo mais poderoso que o Perl e mais orientado a objetos que o Python;
  - Assim, Yukihiro 'Matz' Matsumoto, aos 28 anos, criou o Ruby em 1993;

- Aumento da capacidade de projetar novas linguagens;
  - Certo estudante queria algo mais poderoso que o Perl e mais orientado a objetos que o Python;
  - Assim, Yukihiro 'Matz' Matsumoto, aos 28 anos, criou o Ruby em 1993;
- Avanço global da computação
  - Nem sempre o popular é o melhor

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Aplicações Científicas

- Primeira aplicação da computação (Década de 40);
- Possuem estrutura de dados simples;
- Exigem um grande número de computações aritméticas;
- Primeira Linguagem: Fortran;

Aplicações Comerciais

- Início na década de 50;
- Devem ter facilidades de produzir relatórios;
- Houve pouca pesquisa nesse tipo de linguagem;
- Primeira Linguagem: COBOL;

Inteligência Artificial

- Utiliza computação simbólica;
- Primeira linguagem: LISP (1965);
- Na década de 70 surgiu a programação lógica;
- Principal linguagem: Prolog;

Programação de Sistemas

- Sistemas básicos para o funcionamento do computador;
- A linguagem deve ter rápida execução;
- Deve oferecer, também, recursos de baixo nível;
- Alguns fabricantes desenvolveram suas próprias linguagens;
- O UNIX foi escrito em C;

Linguagens de Scripting

- Comandos são inseridos em arquivos para serem executados;
- Primeira Linguagem: Perl;
- Com o advento da WWW esse tipo de linguagem ganhou muita popularidade;
- Exemplo de Linguagem: Java Script;

Linguagens de Propósitos Especiais

- Linguagens para solucionar problemas específicos;
- GPSS: simulação de sistemas;
- APT: usada para controle de máquinas;

Linguagens de Propósitos Gerais

- Linguagens para solucionar problemas em diversos cenários;

Linguagens de Propósitos Gerais

- Linguagens para solucionar problemas em diversos cenários;
- Java é utilizado para aplicações comerciais e 'científicas';

Linguagens de Propósitos Gerais

- Linguagens para solucionar problemas em diversos cenários;
- Java é utilizado para aplicações comerciais e 'científicas';
- 'Python sempre que pudermos, C++ se necessário'. Alex Martelli, Líder Técnico do Google;

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Legibilidade

- Quão facilmente um programa pode ser lido e entendido;

Legibilidade

- Quão facilmente um programa pode ser lido e entendido;
- Escrita de código X Manutenção de Software (Década de 70);

Legibilidade

- Quão facilmente um programa pode ser lido e entendido;
- Escrita de código X Manutenção de Software (Década de 70);
- Deve-se levar em conta o domínio do problema;
  - Maratona de programação;
  - Google Code Jam;
  - The International Obfuscated C Code Contest;

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender
- Multiplicidade: mais de uma maneira de realizar uma operação particular;
  - cont=cont+1;

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender
- Multiplicidade: mais de uma maneira de realizar uma operação particular;
  - cont=cont+1;
  - cont++;

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender
- Multiplicidade: mais de uma maneira de realizar uma operação particular;
  - cont=cont+1;
  - cont++;
  - ++cont;

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender
- Multiplicidade: mais de uma maneira de realizar uma operação particular;
  - cont=cont+1;
  - cont++;
  - ++cont;
  - cont+=1;

Legibilidade

Simplicidade global

- Leva em conta a quantidade de componentes da linguagem;
  - Linguagens com um pequeno número de componentes básicos são mais fáceis de aprender
- Multiplicidade: mais de uma maneira de realizar uma operação particular;
  - cont=cont+1;
  - cont++;
  - ++cont;
  - cont+=1;
- Problema: um programador pode aprender somente um subconjunto de componentes;

Legibilidade

Ortogonalidade

- Combinação de um conjunto relativamente pequeno de construções primitivas;
  - Os chutes geralmente estão certos;

Legibilidade

Ortogonalidade

- Combinação de um conjunto relativamente pequeno de construções primitivas;
  - Os chutes geralmente estão certos;
- A falta de ortogonalidade acarreta exceções à linguagem;
  - Em C, pode-se retornar um tipo primitivo, mas não um array, somente um ponteiro para [0];

Legibilidade

Ortogonalidade

- Combinação de um conjunto relativamente pequeno de construções primitivas;
  - Os chutes geralmente estão certos;
- A falta de ortogonalidade acarreta exceções à linguagem;
  - Em C, pode-se retornar um tipo primitivo, mas não um array, somente um ponteiro para [0];
- Ortogonalidade leva a simplicidade;

Legibilidade

Ortogonalidade

- Combinação de um conjunto relativamente pequeno de construções primitivas;
  - Os chutes geralmente estão certos;
- A falta de ortogonalidade acarreta exceções à linguagem;
  - Em C, pode-se retornar um tipo primitivo, mas não um array, somente um ponteiro para [0];
- Ortogonalidade leva a simplicidade;
- Muita ortogonalidade pode causar problemas, pois permite instruções extremamente complexas;
  - Em C, int num $= \sim (1 \ll 31)$;

Legibilidade

Instruções de controle

- Instrução goto;

Legibilidade

Instruções de controle

- Instrução goto;
- Atualmente:
  - Instruções de decisão;
  - Instruções de repetição;

Legibilidade

Instruções de controle

- Instrução goto;
- Atualmente:
  - Instruções de decisão;
  - Instruções de repetição;
- A instrução switch para evitar if aninhados;

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
    - timeOut = 1 (significado não claro);

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
    - timeOut = 1 (significado não claro);
    - timeOut = true (significado claro);

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
    - timeOut = 1 (significado não claro);
    - timeOut = true (significado claro);
- Inteiros com precisão arbitrária;

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
    - timeOut = 1 (significado não claro);
    - timeOut = true (significado claro);
- Inteiros com precisão arbitrária;
- Números complexo;

Legibilidade

Tipos de dados e estruturas

- A presença de facilidades adequadas para definir tipos de dados e estruturas de dados;
- O significado dos tipos devem ser claros;
  - Suponha que em uma linguagem não exista um tipo de dado booleano e um tipo numérico seja usado para substituí-lo:
    - timeOut = 1 (significado não claro);
    - timeOut = true (significado claro);
- Inteiros com precisão arbitrária;
- Números complexo;
- Inteiros com representação decimal.

Legibilidade

Considerações sobre a sintaxe

- Formas identificadoras;
  - Restringir os identificadores a tamanhos muito pequenos prejudica a legibilidade. Ex.: int x,x1,x2;

Legibilidade

Considerações sobre a sintaxe

- Formas identificadoras;
  - Restringir os identificadores a tamanhos muito pequenos prejudica a legibilidade. Ex.: int x,x1,x2;
- Palavras especiais;
  - Formas das palavras especiais de uma linguagem. Ex.: while, class, for e begin-end;

Legibilidade

Considerações sobre a sintaxe

- Formas identificadoras;
  - Restringir os identificadores a tamanhos muito pequenos prejudica a legibilidade. Ex.: int x,x1,x2;
- Palavras especiais;
  - Formas das palavras especiais de uma linguagem. Ex.: while, class, for e begin-end;
  - Palavras especiais de uma linguagem podem ser usadas como nomes de variáveis?

Legibilidade

Considerações sobre a sintaxe

- Formas identificadoras;
  - Restringir os identificadores a tamanhos muito pequenos prejudica a legibilidade. Ex.: int x,x1,x2;
- Palavras especiais;
  - Formas das palavras especiais de uma linguagem. Ex.: while, class, for e begin-end;
  - Palavras especiais de uma linguagem podem ser usadas como nomes de variáveis?
- Forma e significado;
  - Projetar instruções de forma que sua aparência indique sua finalidade. Ex.: if, while;

Capacidade de Escrita (Writability)

- Medida de facilidade para se escrever programas;

Capacidade de Escrita (Writability)

- Medida de facilidade para se escrever programas;
- A legibilidade influencia a capacidade de escrita;

Capacidade de Escrita

- Simplicidade e ortogonalidade;
  - Muita ortogonalidade pode levar a:

Capacidade de Escrita

- Simplicidade e ortogonalidade;
  - Muita ortogonalidade pode levar a:
    - Uso inadequado de recursos;
    - Desuso de recursos;
    - Uso de recursos desconhecidos por acidente;

Capacidade de Escrita

- Suporte para abstração;
  - A capacidade de definir e de usar estruturas ou operações complexas de maneira que permita ignorar muitos dos detalhes;
- Expressividade;
  - Um conjunto relativamente conveniente de maneiras de especificar operadores;
  - Exemplos:
    - count++ é mais conveniente do que count = count + 1;
    - A inclusão do for em muitas linguagens modernas;

Confiabilidade

- O programa deve funcionar de acordo com as suas especificações sob todas as condições;

Confiabilidade

- O programa deve funcionar de acordo com as suas especificações sob todas as condições;
- Verificação de Tipos;

Confiabilidade

- O programa deve funcionar de acordo com as suas especificações sob todas as condições;
- Verificação de Tipos;
- Manipulação de Exceções;

Confiabilidade

Verificação de Tipos

- Verificação se existem erros de tipo em um programa;
- Verificação Estática (Tempo de Compilação);
- Verificação Dinâmica (Tempo de Execução);
- Quanto mais cedo os erros forem encontrados, melhor!

Confiabilidade

Manipulação de Exceções

- Tratamento de Exceções;
- Identificar erros em tempo de execução;
- Tomar medidas corretivas;
- Prosseguir a execução do programa;

Confiabilidade

Apelidos

- Dois nomes se referenciando à mesma posição de memória;
- Exemplo: ponteiros apontando para a mesma célula;

Confiabilidade

Apelidos

- Dois nomes se referenciando à mesma posição de memória;
- Exemplo: ponteiros apontando para a mesma célula;
- É um recurso perigoso;

Confiabilidade

Legibilidade e Capacidade de Escrita

- Programas fáceis de ler e de escrever são mais confiáveis;
- Facilidade de Manutenção;

Confiabilidade

Legibilidade e Capacidade de Escrita

- Programas fáceis de ler e de escrever são mais confiáveis;
- Facilidade de Manutenção;
- Uma linguagem que não suporta maneiras naturais de expressar os algoritmos usará, necessariamente, abordagens não-naturais (gambiarras!);

Custo

- Treinamento dos programadores;
- Custo de escrever programas;
- Custo de compilação X Custo de Execução;
- Existência de um sistemas de implementação;
- Custo de manutenção dos programas;

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Influências sobre o projeto de uma linguagem

- Arquitetura do Computador;
  - É um fator crucial sobre o projeto de uma linguagem;
  - A maioria é baseada na Arquitetura de von Neumann (1945);
    - Dados e Programas são armazenados na mesma memória;
    - Possui uma UCP que executa as instruções;
    - Os dados devem ser canalizados (piped) da memória para UCP (vice-versa);
    - Gargalo de von Neumann;

Influências sobre o projeto de uma linguagem

- Arquitetura do Computador;
  - É um fator crucial sobre o projeto de uma linguagem;
  - A maioria é baseada na Arquitetura de von Neumann (1945);
    - Dados e Programas são armazenados na mesma memória;
    - Possui uma UCP que executa as instruções;
    - Os dados devem ser canalizados (piped) da memória para UCP (vice-versa);
    - Gargalo de von Neumann;
- Metodologias de programação;
  - Novas metodologias de desenvolvimento de software (exemplo, desenvolvimento orientado a objetos) levam a novos paradigmas de programação, e novas linguagens de programação;

Influências sobre o projeto de uma linguagem

Metodologias de programação

- Década de 1950 e início da década de 1960;
  - Aplicações simples; preocupações com a eficiência da máquina;

Influências sobre o projeto de uma linguagem

Metodologias de programação

- Década de 1950 e início da década de 1960;
  - Aplicações simples; preocupações com a eficiência da máquina;
- Final da década de 1960;
  - A eficiência das pessoas se tornou um ponto importante;
  - Legibilidade, melhores estruturas de controle;
  - Programação estruturada;

Influências sobre o projeto de uma linguagem

Metodologias de programação

- Década de 1950 e início da década de 1960;
  - Aplicações simples; preocupações com a eficiência da máquina;
- Final da década de 1960;
  - A eficiência das pessoas se tornou um ponto importante;
  - Legibilidade, melhores estruturas de controle;
  - Programação estruturada;
- Final da década de 1970;
  - Orientado para processo $\rightarrow$ orientada a dados;
  - Abstração de dados;

Influências sobre o projeto de uma linguagem

Metodologias de programação

- Década de 1950 e início da década de 1960;
  - Aplicações simples; preocupações com a eficiência da máquina;
- Final da década de 1960;
  - A eficiência das pessoas se tornou um ponto importante;
  - Legibilidade, melhores estruturas de controle;
  - Programação estruturada;
- Final da década de 1970;
  - Orientado para processo $\rightarrow$ orientada a dados;
  - Abstração de dados;
- Meados da década de 1980;
  - Programação orientada a objetos;
  - Abstração de dados + herança + polimorfismo;

Influências sobre o projeto de uma linguagem

Metodologias de programação

- Década de 1950 e início da década de 1960;
  - Aplicações simples; preocupações com a eficiência da máquina;
- Final da década de 1960;
  - A eficiência das pessoas se tornou um ponto importante;
  - Legibilidade, melhores estruturas de controle;
  - Programação estruturada;
- Final da década de 1970;
  - Orientado para processo $\rightarrow$ orientada a dados;
  - Abstração de dados;
- Meados da década de 1980;
  - Programação orientada a objetos;
  - Abstração de dados + herança + polimorfismo;
- Década de 90 - Atualmente;
  - Adicionam-se maior interatividade (internet, web);
  - Novos paradigmas (orientado a aspectos, componentes);

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Categorias de linguages

- Imperativas (C, Pascal, Fortran, Algol, Cobol);
  - Características centrais: variáveis, estruturas de atribuição e iteração;

Categorias de linguages

- Imperativas (C, Pascal, Fortran, Algol, Cobol);
  - Características centrais: variáveis, estruturas de atribuição e iteração;
- Funcionais (Lisp, ML, Scheme);
  - Principal maneira de computar é através da aplicação de funções para os parâmetros fornecidos. EX. mult(a,(som(c,d)));

Categorias de linguages

- Imperativas (C, Pascal, Fortran, Algol, Cobol);
  - Características centrais: variáveis, estruturas de atribuição e iteração;
- Funcionais (Lisp, ML, Scheme);
  - Principal maneira de computar é através da aplicação de funções para os parâmetros fornecidos. EX. mult(a,(som(c,d)));
- Lógicas (Prolog, Datalog);
  - Baseado em regras (regras são especificadas sem ordem pré-determinada);

Categorias de linguages

- Imperativas (C, Pascal, Fortran, Algol, Cobol);
  - Características centrais: variáveis, estruturas de atribuição e iteração;
- Funcionais (Lisp, ML, Scheme);
  - Principal maneira de computar é através da aplicação de funções para os parâmetros fornecidos. EX. mult(a,(som(c,d)));
- Lógicas (Prolog, Datalog);
  - Baseado em regras (regras são especificadas sem ordem pré-determinada);
- Orientadas a Objetos (Smalltalk, C++, Eifel, Java, Python, Ruby);
  - Abstração de dados, herança;

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Custo/benefício no projeto da linguagem

- São muitos critérios conflitantes;
  - Confiabilidade e Custo de Execução (Exemplo: Matrizes em Ada e C);
  - Expresividade e Legibilidade;
  - Flexibilidade e Segurança;

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Métodos de implementação

- Compilação;
  - Programas são traduzidos para linguagem de máquina;
- Interpretação pura;
  - Programas são interpretados por outro programa conhecido como interpretador;
- Sistemas de Implementação Híbridos;
  - Um meio-termo entre compiladores e interpretadores puros;

Compilação

- Traduz programas em alto-nível (linguagem fonte) em código de máquina (linguagem de máquina);

Compilação

- Traduz programas em alto-nível (linguagem fonte) em código de máquina (linguagem de máquina);
- Tradução lenta, execução rápida;
- O processo de compilação possui várias fases:
  - Análise léxica - converte caracteres de um programa fonte em unidades léxicas (análise dos tokens). Ex.: for, whille;

Compilação

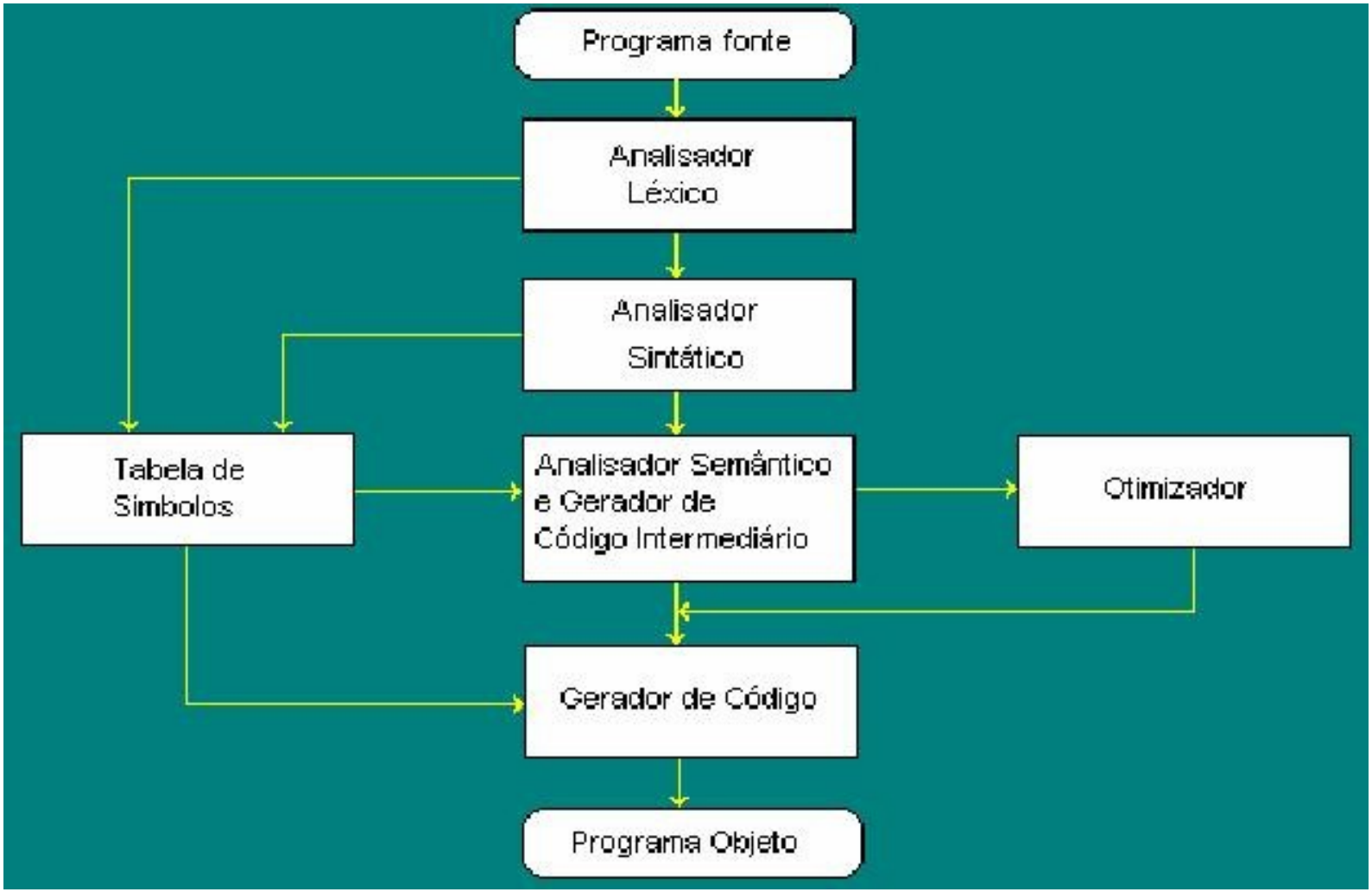
- Traduz programas em alto-nível (linguagem fonte) em código de máquina (linguagem de máquina);
- Tradução lenta, execução rápida;
- O processo de compilação possui várias fases:
  - Análise léxica - converte caracteres de um programa fonte em unidades léxicas (análise dos tokens). Ex.: for, whille;
  - Análise sintática - Transforma unidades léxicas em parse trees, as quais representam a estrutura sintática do programa;

Compilação

- Traduz programas em alto-nível (linguagem fonte) em código de máquina (linguagem de máquina);
- Tradução lenta, execução rápida;
- O processo de compilação possui várias fases:
  - Análise léxica - converte caracteres de um programa fonte em unidades léxicas (análise dos tokens). Ex.: for, whille;
  - Análise sintática - Transforma unidades léxicas em parse trees, as quais representam a estrutura sintática do programa;
  - Análise semântica - Tentar entender o 'significado' do programa. Ex.: verificação no uso de tipos, escopo, atribuições;

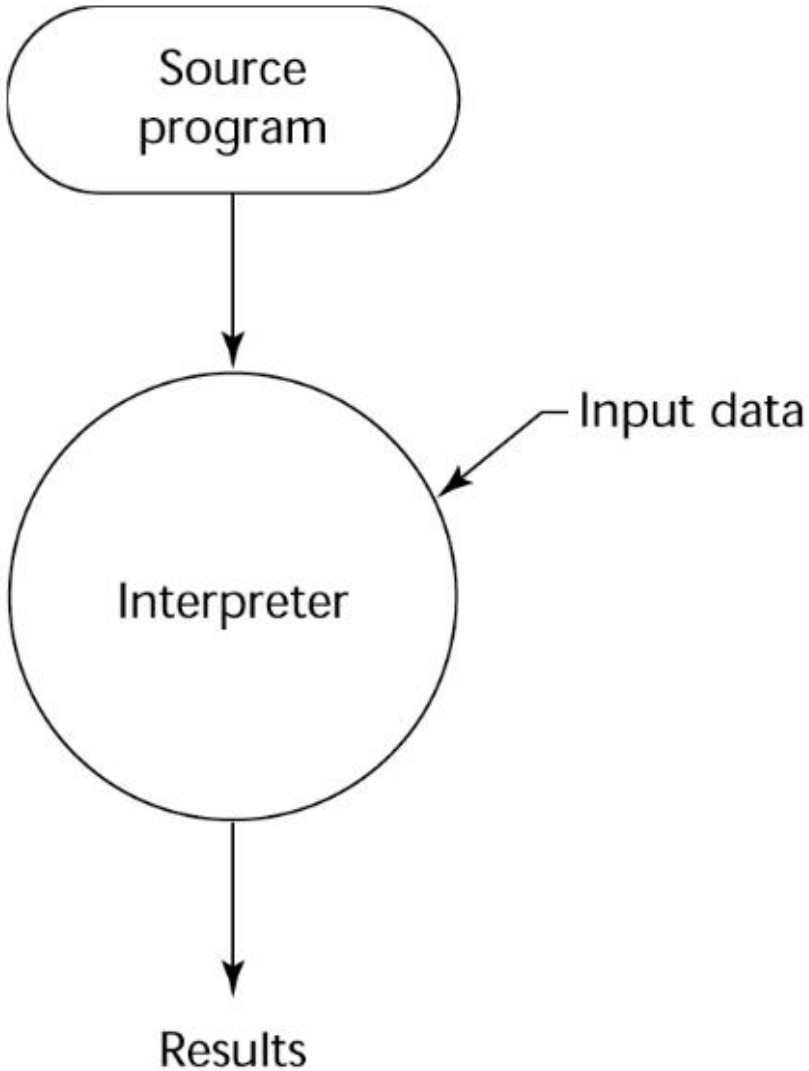
Compilação

- Traduz programas em alto-nível (linguagem fonte) em código de máquina (linguagem de máquina);
- Tradução lenta, execução rápida;
- O processo de compilação possui várias fases:
  - Análise léxica - converte caracteres de um programa fonte em unidades léxicas (análise dos tokens). Ex.: for, whille;
  - Análise sintática - Transforma unidades léxicas em parse trees, as quais representam a estrutura sintática do programa;
  - Análise semântica - Tentar entender o 'significado' do programa. Ex.: verificação no uso de tipos, escopo, atribuições;
  - Geração de código - Código otimizado intermediário e geração de código de máquina;


Programa fonte
Analisador Léxico
Analisador Sintático
Tabela de Simbolos
Analisador Semântico e Gerador de Código Intermediário
Otimizador
Gerador de Código
Programa Objeto

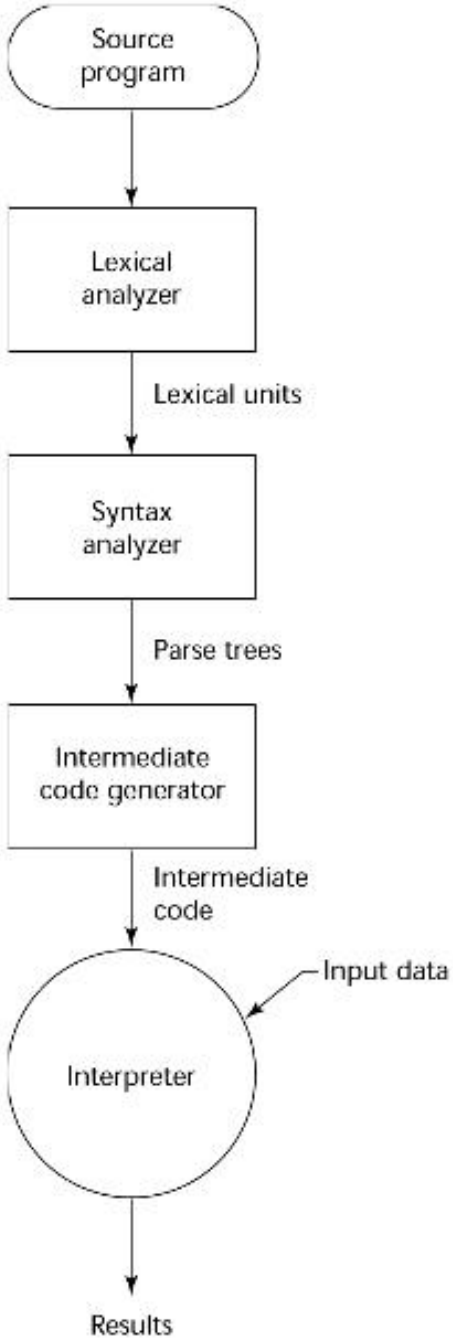
Interpretação Pura

- Sem tradução;
- Fácil implementação de programas (erros de execução podem ser facilmente e rapidamente mostrados);
- Execução lenta (de 10 a 100 vezes mais lenta do que programas compilados);
- Geralmente requer mais espaço;
- Cada vez mais raro em linguagens de alto-nível;


Source program
Input data
Interpreter
Results

Sistemas de Implementação Híbridos

- Um meio-termo entre compilador e interpretador puro;
- Gera-se o código intermediário (compilação);
- Interpreta-se esse código intermediário (interpretação);
- Processo mais rápido do que a Interpretação pura;
- Garante maior portabilidade;
- Exemplos: Perl, Java


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Intermediate code generator
Intermediate code
Input data
Interpreter
Results

Agenda

1. Razões para estudar os conceitos de linguagens de programação
2. Domínios de programação
3. Critérios de avaliação de linguagens
4. Influências sobre o projeto de uma linguagem
5. Categorias de linguagens
6. Custo/benefício no projeto da linguagem
7. Métodos de implementação
8. Ambientes de programação

Agenda

Sistemas de Implementação Híbridos

- Conjunto de Ferramentas utilizadas para construção de softwares;
- Principais componentes:
  - Sistema de arquivos;
  - Editor de texto;
  - Compilador;
  - Linkeditor;
- Exemplos: Eclipse, Netbeans, Code::Blocks, Visual Studio, DEV-C++, Lazarus etc;