Conceitos de Linguagens de Programação

A T2Ti

Apresentação

A T2Ti nasce do sonho de três colegas que trabalhavam no maior banco da américa latina.

Tudo começa em 2007 com o lançamento do curso Java Starter. Logo depois veio o Siscom Java Desktop seguido de outros treinamentos.

Desde então a Equipe T2Ti se esforça para produzir material de qualidade que possa formar profissionais para o mercado, ensinando como desenvolver sistemas de pequeno, médio e grande porte.

Um dos maiores sucessos da Equipe T2Ti foi o Projeto T2Ti ERP que reuniu milhares de profissionais num treinamento dinâmico onde o participante aprendia na prática como desenvolver um ERP desde o levantamento de requisitos. Foi através desse treinamento que centenas de desenvolvedores iniciaram seu negócio próprio e/ou entraram no mercado de trabalho.

Em 2010 a T2Ti lança sua primeira aplicação para produção, o Controle Financeiro Pessoal. O sucesso foi tanto que saiu até em matéria no site Exame, ficando entre os 10 aplicativos mais baixados da semana.

Começa então a era de desenvolvimento de sistemas para alguns clientes exclusivos, pois o foco ainda era em desenvolvimento de treinamentos. A T2Ti desenvolve sistemas para o mercado nacional e internacional.

Atualmente a T2Ti se concentra nas duas vertentes: desenvolver sistemas e produzir treinamentos.

_____

Sumário

Conceitos de Linguagens de Programação

Linguagens de Programação
Introdução; Resumo Histórico; Interpretação e Compilação – Resumo; Conceitos Iniciais: Programação Estruturada, Programação Linear, Programação Orientada a Objetos.

Aspectos Preliminares
Critérios de Avaliação de Linguagens; Classificação.

Evolução
Introdução; Genealogia; Antes de 1940; A Década de 1940; As Décadas de 1950 e 1960; 1967 a 1978 | Paradigmas Fundamentais; Anos 80 | Consolidação, Módulos, Performance; Anos 90 | A Era da Internet; Tendências.

Tópicos Importantes
Variáveis; Tipos de Dados; Estruturas de Controle; Subprograma ou Sub-rotina; Passagem de Parâmetros.

Índice Tiobe
Apresentação.

Linguagens de Programação

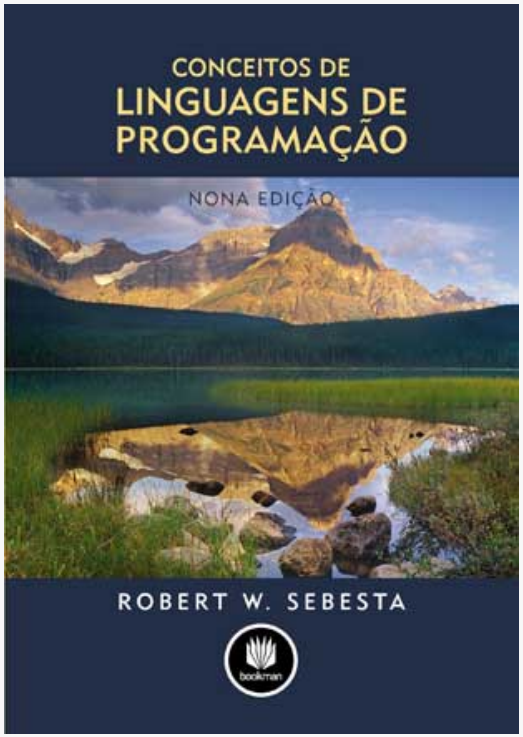
Introdução

Muitos desenvolvedores que estão no mercado, aprenderam a programar em cursos técnicos e já começaram a desenvolver soluções. Alguns jamais fizeram uma faculdade. Isso é mais comum do que parece.

No passado era ainda mais natural. Existiam vários desenvolvedores Clipper que faziam sistemas de locadora, farmácia, etc. Com o tempo veio o Delphi e o Visual Basic e muitos desses desenvolvedores migraram para essas linguagens/ferramentas.

Mas falta alguma coisa para parte desses colegas: os conceitos das linguagens de programação. Tais conceitos podem ser assimilados com a leitura do livro do Sebesta, que tem exatamente esse título. Faremos uma síntese dos conceitos. Eles são importantes para que não haja dificuldades na compreensão dos temas abordados durante o estudo do Projeto T2Ti ERP.

Quem desejar se aprofundar no tema, deverá adquirir o livro do Sebesta.



Linguagens de Programação

Introdução

Uma linguagem de programação é um método padronizado para comunicar instruções para um computador. É um conjunto de regras sintáticas e semânticas usadas para definir um programa de computador.

Permitem que um programador especifique precisamente sobre quais dados um computador vai atuar, como estes dados serão armazenados ou transmitidos e quais ações devem ser tomadas sob várias circunstâncias.

As linguagens de programação podem ser usadas para expressar algoritmos com precisão.

O conjunto de palavras, compostas de acordo com essas regras da linguagem, constituem o código fonte de um software. Esse código fonte é depois traduzido para código de máquina, que é executado pelo processador.

Uma das principais metas das linguagens de programação é que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente (código de máquina).

Dessa forma, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos.

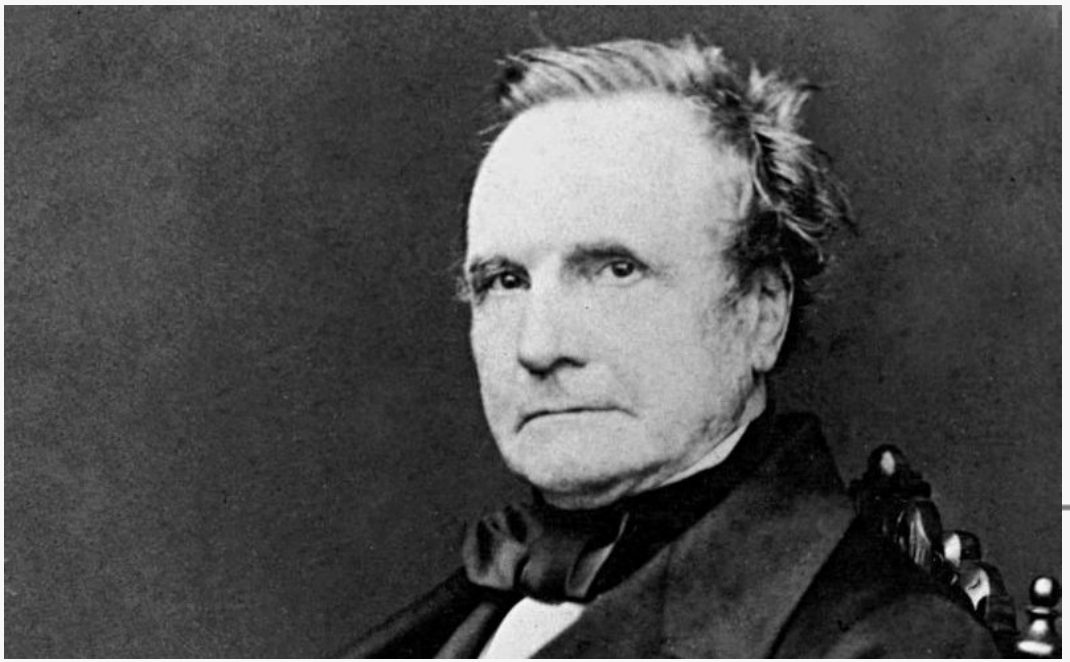
Linguagens de programação são ferramentas importantes para que programadores e engenheiros de software possam escrever programas mais organizados e com maior rapidez.

Linguagens de Programação

Resumo Histórico

O primeiro trabalho de linguagem de programação foi criado por Ada Lovelace (vista na imagem ao lado - 1840), grande amiga de Charles Babbage.

O projeto da primeira calculadora mecânica programável foi idealizado por Charles Babbage (visto na imagem abaixo - 1860) que, após gastar fortunas e um longo tempo, não conseguiu concretizar o projeto. A linguagem de programação ADA foi batizada em homenagem a esta primeira programadora.



Resumo Histórico

Uma das primeiras linguagens de programação para computadores foi provavelmente Plankalkül, criada por Konrad Zuse na Alemanha Nazista, mas que teve pouco ou nenhum impacto no futuro das linguagens de programação.

O primeiro compilador foi escrito por Grace Hopper, em 1952, para a linguagem de programação A-0.

A primeira linguagem de programação de alto nível amplamente usada foi o Fortran, criada em 1954. Em 1957 foi criada a B-0, sucessora da A-0, que daria origem a Flow-Matic (1958), antecessor imediato de COBOL, de 1959.

O COBOL foi uma linguagem de ampla aceitação para uso comercial. A linguagem ALGOL foi criada em 1958-1960. O ALGOL-60 teve grande influência no projeto de muitas linguagens posteriores.

A linguagem Lisp foi criada em 1958 e se tornou amplamente utilizada na pesquisa na área de ciência da computação mais proeminentemente na área de Inteligência Artificial.

Outra linguagem relacionada ao campo da IA que surge em 1972 é a linguagem Prolog, uma linguagem do paradigma lógico.

A orientação a objetos é outro marco importante na história das linguagens de programação. A linguagem Simula 67 introduz o conceito de classes. A linguagem Smalltalk expande o conceito de classes e se torna a primeira linguagem de programação que oferecia suporte completo à programação orientada a objetos. A linguagem C++ (originalmente conhecida como C com classes) popularizou a orientação a objetos.

Interpretação e Compilação – Resumo

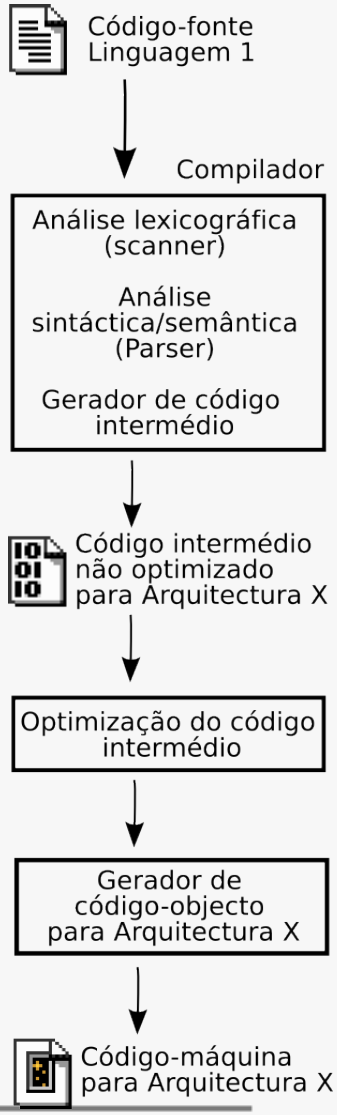
Uma linguagem de programação pode ser convertida, ou traduzida, em código de máquina por compilação (processo visto na imagem ao lado) ou interpretada por um processo denominado interpretação. Em ambas ocorre a tradução do código fonte para código de máquina.

Se o método utilizado traduz todo o texto do programa (também chamado de código), para só depois executar o programa, então diz-se que o programa foi compilado e que o mecanismo utilizado para a tradução é um compilador (que por sua vez nada mais é do que um programa).

A versão compilada do programa tipicamente é armazenada, de forma que o programa pode ser executado um número indefinido de vezes sem que seja necessária nova compilação, o que compensa o tempo gasto na compilação. Isso acontece com linguagens como Pascal e C.

Se o texto do programa é executado à medida que vai sendo traduzido, como em JavaScript, BASIC, Python, Perl ou PHP, num processo de tradução de trechos seguidos de sua execução imediata, então diz-se que o programa foi interpretado e que o mecanismo utilizado para a tradução é um interpretador.

Programas interpretados são geralmente mais lentos do que os compilados, mas são também geralmente mais flexíveis, já que podem interagir com o ambiente mais facilmente.



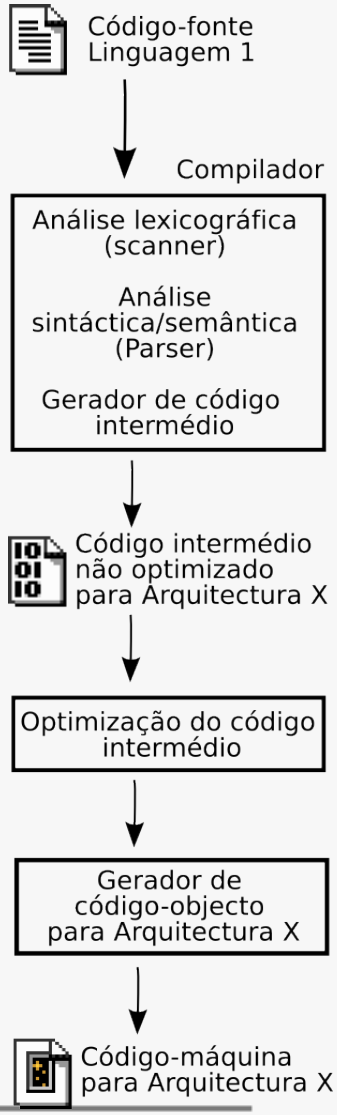
Interpretação e Compilação – Resumo

Embora haja essa distinção entre linguagens interpretadas e compiladas, as coisas nem sempre são tão simples. Há linguagens compiladas para um código de máquina virtual (sendo esta máquina virtual apenas mais um software), como Java (compila para a plataforma Java) e C# (compila para a plataforma CLI - .NET).

E também há outras formas de interpretar, onde os códigos fontes, no lugar de serem interpretados linha-a-linha, têm blocos "compilados" para a memória, de acordo com as necessidades, o que aumenta a performance dos programas quando os mesmos módulos são chamados várias vezes, técnica esta conhecida como JIT.

Como exemplo, podemos citar a linguagem Java. Nela, um compilador traduz o código java para o código intermediário (e portável) da JVM. As JVMs originais interpretavam esse código, de acordo com o código de máquina do computador hospedeiro, porém atualmente elas compilam, segundo a técnica JIT o código JVM para código hospedeiro.

A tradução é tipicamente feita em várias fases, sendo as mais comuns a análise léxica, a análise sintática (ou *parsing*), a geração de código e a otimização. Em compiladores também é comum a geração de código intermediário.



Conceitos Iniciais | Programação Estruturada

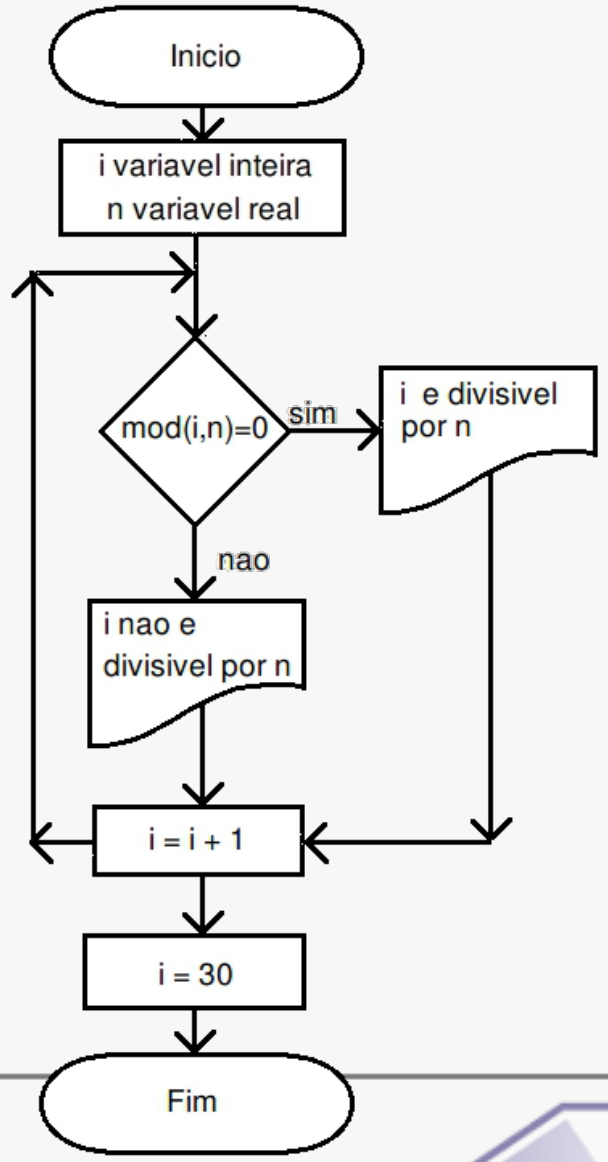
É uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência, decisão e iteração.

Um dos primeiros a preconizar a programação estruturada foi Haskell B. Curry.

Tendo, na prática, sido transformada na programação modular, a programação estruturada orienta os programadores para a criação de estruturas simples em seus programas, usando as sub-rotinas e as funções.

Foi a forma dominante na criação de software entre a programação linear e a programação orientada por objetos.

Apesar de ter sido sucedida pela programação orientada por objetos, pode-se dizer que a programação estruturada ainda é marcantemente influente, uma vez que grande parte das pessoas ainda aprendem programação através dela.



Linguagens de Programação

Conceitos Iniciais | Programação Linear

Em matemática, problemas de Programação Linear são problemas de otimização nos quais a função objetivo e as restrições são todas lineares. Muitos problemas práticos em pesquisa operacional podem ser expressos como problemas de programação linear.

Certos casos especiais de programação linear, tais como problemas de network flow e problemas de multicommodity flow são considerados importantes o suficiente para que se tenha gerado muita pesquisa em algoritmos especializados para suas soluções.

Vários algoritmos para outros tipos de problemas de otimização funcionam resolvendo problemas de PL como sub-problemas. Historicamente, ideias da programação linear inspiraram muitos dos conceitos centrais de teoria da otimização, tais como dualidade, decomposição, e a importância da convexidade e suas generalizações.

Pesquisa operacional é o uso de modelos matemáticos, estatística e algoritmos para ajudar a tomada de decisões.

Empresas como FedEx, American Airlines, General Eletric, Xerox, Vale, Petrobras, Usiminas, Pirelli, Bancos de Investimento, entre outras, tem confiado, cada vez mais, suas operações estratégicas em modelos matemáticos de Pesquisa Operacional.

Conceitos Iniciais | Programação Orientada a Objetos

Orientação a objetos, também conhecida como Programação Orientada a Objetos (POO), ou ainda em inglês Object-Oriented Programming (OOP) é um paradigma de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

O extensivo uso de objetos, particularmente em conjunção com o mecanismo de herança, caracteriza o estilo de programação orientada a objetos.

Na qualidade de método de modelagem, é tida como a melhor estratégia, e mais natural, para se eliminar o "gap semântico", dificuldade recorrente no processo de modelar o mundo real, no domínio do problema, em um conjunto de componentes de software que seja o mais fiel na sua representação deste domínio.

Facilitaria a comunicação do profissional modelador e do usuário da área alvo, na medida em que a correlação da simbologia e conceitos abstratos do mundo real e da ferramenta de modelagem (conceitos, terminologia, símbolos, grafismo e estratégias) fosse a mais óbvia, natural e exata possível.

A análise e projeto orientados a objetos tem como meta identificar o melhor conjunto de objetos para descrever um sistema de software.

O funcionamento deste sistema se dá através do relacionamento e troca de mensagens entre estes objetos. Na programação orientada a objetos, implementa-se um conjunto de classes que definem os objetos presentes no sistema de software. Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

Critérios de Avaliação de Linguagens

Podemos separar os critérios de avalização de linguagens em quatro itens:

- Legibilidade
- Capacidade (ou facilidade) de Escrita (Writability)
- Confiabilidade
- Custo

Legibilidade

A legibilidade é uma qualidade que determina a facilidade de leitura de alguma coisa. Características que contribuem para legibilidade numa linguagem de programação:

- Simplicidade: poucos componentes básicos, cuidado com a multiplicidade de recursos, overloading inteligente de operadores. Não é bom ter muitos componentes básicos, assim como é ruim ter mais de uma maneira de realizar uma operação.
- Ortogonalidade: conjunto consistente de regras para combinar construções primitivas, com poucas exceções. Torna as linguagens fáceis de leitura e aprendizagem. O significado de um recurso da linguagem é independente do contexto. Um conjunto relativamente pequeno de construções primitivas podem ser combinadas em um número relativamente pequeno de maneiras para construir as estruturas de dados e controle da linguagem. Qualquer combinação possível das construções primitivas é aceita.
- Instruções de controle.
- Tipos e estruturas de dados;
- Sintaxe: tamanho dos identificadores. Uso de palavras reservadas.

Critérios de Avaliação de Linguagens

Capacidade (ou facilidade) de Escrita (Writability)

A legibilidade é uma qualidade que determina a facilidade de leitura de alguma coisa. Características que contribuem para legibilidade numa linguagem de programação:

- Simplicidade: grande número de construções diferentes leva a dificuldades para escrever programas.
- Suporte para abstração: capacidade de definir estruturas/operações complexas ignorando detalhes. Abstração pode ser em dados e em código.
- Expressividade: adequação das formas de especificar computações.

Confiabilidade

No geral, confiabilidade é a capacidade de uma pessoa ou sistema de realizar e manter seu funcionamento em circunstâncias de rotina, bem como em circunstâncias inesperadas.

Em termos de linguagens de programação, temos as seguintes características:

- Verificação de tipos.
- Tratamento de exceções.
- Apelidos (aliasing).
- Pouca legibilidade ou pouca facilidade de escrita tendem a gerar programas pouco confiáveis.

Aspectos Preliminares

Critérios de Avaliação de Linguagens

Custo

Para determinar o custo final de uma linguagem de programação, devemos levar em consideração alguns fatores.

- Treinamento: quanto maior a complexidade e quanto mais recursos contém a linguagem maior o grau de dificuldade de aprendizado.
- Programação: está ligado aos fatores de simplicidade. Quanto menor o numero de componentes básicos, mais fácil à leitura e a programação do código fonte. Desenvolver um código otimizado com execução mais rápida.
- Testes: testes realizados na linguagem visam confiança. Em sistemas críticos, se houverem falhas, o custo pode tornar-se elevado.
- Manutenção: visa corrigir ou modificar para adicionar/remover recursos. O custo da manutenção mede-se principalmente pelas suas características de legibilidade, quanto mais fácil é escrever o programa, mais fácil torna-se a manutenção. Os fatores mais consideráveis em termos de custos de linguagem são o desenvolvimento do programa, manutenção e confiabilidade, sendo as duas últimas mais onerosas.
- Evolução: é complicado prever a evolução da linguagem, mesmo da aplicação desenvolvida pela linguagem, quando notamos a grande variedade de dispositivos e tecnologias que surgem frequentemente, as quais podemos implementar em nossas aplicações. Um fator muito importante diz respeito justamente à capacidade de implementações: a portabilidade. Deve ser avaliado o que determina a faixa de utilização da linguagem e encontrar boa definição na documentação sobre a que se propõe a linguagem.

Classificação

Existem várias maneiras de classificar as linguagens de programação. A Association for Computing Machinery (Associação para Maquinaria da Computação) ou ACM, foi fundada em 1947 como a primeira sociedade cientifica e educacional dedicada a computação. Ela mantém um sistema de classificação com os seguintes subitens:

- Linguagens aplicativas, ou de aplicação.
- Linguagens concorrentes, distribuídas e paralelas.
- Linguagens de fluxo de dados.
- Linguagens de projeto.
- Linguagens extensíveis.
- Linguagens de montagem e de macro.
- Linguagens de microprogramação.
- Linguagens não determinísticas.
- Linguagens não procedurais.
- Linguagens orientadas a objeto.
- Linguagens de aplicação especializada.
- Linguagens de altíssimo nível.

É possível ainda classificar as linguagens quanto ao paradigma. Paradigma é um conceito das ciências e da epistemologia (a teoria do conhecimento) que define um exemplo típico ou modelo de algo.

Diferentes linguagens de programação podem ser agrupadas segundo o paradigma que seguem para abordar a sua sintaxe e semântica. Os paradigmas se dividem em dois grandes grupos: imperativo e declarativo.

Aspectos Preliminares

Classificação

Paradigmas Imperativos

Os paradigmas imperativos são aqueles que facilitam a computação por meio de mudanças de estado. Se dividem em:

- Procedural: neste paradigma, os programas são executados através de chamadas sucessivas a procedimentos separados. Exemplos: o Fortran e o BASIC.
- Estruturas de Blocos: a característica marcante deste paradigma são os escopos aninhados. Exemplos: o Algol 60, Pascal e C.
- Orientação a Objetos: este paradigma descreve linguagens que suportam a interação entre objetos. Exemplos: C++, Java, Python e Ruby.
- Computação Distribuída: este paradigma suporta que mais de uma rotina possa executar independentemente. Um exemplo de linguagem deste paradigma é a linguagem Ada.

Paradigmas Declarativos

Os paradigmas declarativos são aqueles nos quais um programa especifica uma relação ou função. Se dividem em:

- Funcional: linguagens deste paradigma não incluem qualquer provisão para atribuição ou dados mutáveis. Na programação funcional, o mapeamento entre os valores de entrada e saída são alcançados mais diretamente. Um programa é uma função (ou grupo de funções), tipicamente constituída de outras funções mais simples. Exemplos de linguagens deste paradigma são as linguagens Lisp, Scheme e Haskell.
- Programação Lógica: este paradigma se baseia na noção de que um programa implementa uma relação ao invés de um mapeamento. Exemplos de linguagens deste paradigma são o Prolog e a linguagem Gödel.

Aspectos Preliminares

Classificação

Podemos continuar a classificação das linguagens de programação quanto a estrutura de tipos:

- Fracamente Tipada: onde o tipo da variável muda dinamicamente conforme a situação. Exemplos: PHP e Smalltalk.
- Fortemente Tipada: onde o tipo da variável, uma vez atribuído, se mantém o mesmo até ser descartada da memória. Exemplos: Java e Ruby.
- Dinamicamente Tipada: onde o tipo da variável é definido em tempo de execução. Exemplos: SNOBOL, APL, Awk, Perl, Python e Ruby.
- Estaticamente Tipada: onde o tipo da variável é definido em tempo de compilação. Exemplos: Java e C.

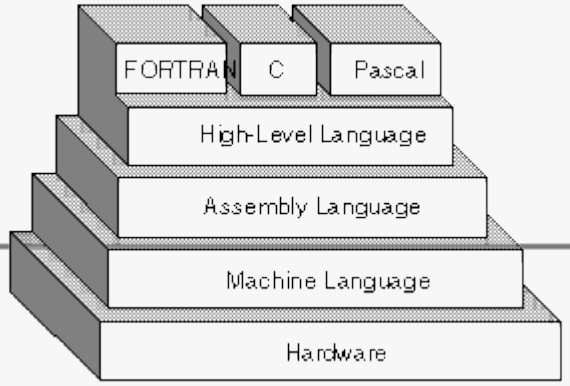
Temos ainda a classificação quanto ao grau de abstração da linguagem:

- Linguagem de Programação de Baixo Nível: cujos símbolos são uma representação direta do código de máquina que será gerado, onde cada comando da linguagem equivale a um "opcode" (código de operação) do processador. Exemplos: Assembly.
- Linguagem de Programação de Médio Nível: possui símbolos que podem ser convertidos diretamente para código de máquina (GOTO, expressões matemáticas, atribuição de variáveis), mas também símbolos complexos que são convertidos por um compilador. Exemplos: C, C+.
- Linguagem de Programação de Alto Nível: composta de símbolos mais complexos, inteligível pelo ser humano e não-executável diretamente pela máquina, no nível da especificação de algoritmos. Exemplos: Pascal, Fortran, ALGOL, Java e SQL.

Classificação

Por fim, podemos classificar as linguagens de programação quanto à geração. No entanto, a classificação das linguagens de programação em gerações é uma questão que apresenta divergências de autor para autor. Segundo MACLENNAN (1999), as linguagens se dividem em cinco gerações com as seguintes características:

- Primeira Geração: são linguagens onde suas estruturas de controle são aparentemente orientadas a máquina. As instruções condicionais não são aninhadas e dependem fortemente de instruções de desvio incondicional como o GOTO. Uma linguagem típica desta geração é a linguagem Fortran.



- Segunda Geração: são linguagens onde as estruturas de controle são estruturadas de forma a minimizar ou dispensar o uso de instruções GOTO. A segunda geração elaborou melhor e generalizou diversas estruturas de controle das linguagens de primeira geração. Uma das grandes contribuições desta geração foi suas estruturas de nomes, que eram hierarquicamente aninhadas. Isto permitiu melhor controle de espaços de nomes e uma eficiente alocação dinâmica de memória. Uma linguagem típica desta geração é o Algol 60.
- Terceira Geração: são linguagens que dão ênfase à simplicidade e eficiência. Uma linguagem típica desta geração é a linguagem Pascal. As estruturas de dados desta geração mostram um deslocamento da máquina para a aplicação. As estruturas de controle são mais simples e eficientes.

Aspectos Preliminares

Classificação

- Quarta Geração: esta geração é essencialmente o sinônimo para linguagens com abstração de dados. A maioria das linguagens desta geração focam na modularização e no encapsulamento. Uma linguagem típica desta geração é a linguagem Ada.
- Quinta Geração: nesta geração, Maclennan agrupa diversos paradigmas como a orientação a objeto e os paradigmas funcional e lógico.

Henri Bal e Dick Grune, já apresentam uma classificação em gerações de forma diferente, enfatizando mais o aspecto da aplicação. São elencadas 6 gerações.

Doris Apleby e Julius J. VandeKopple dividem as linguagens em quatro gerações que coincidem com as quatro primeiras gerações elencadas por Henri Bal e Dick Grune.

- Primeira Geração: linguagem de máquina.
- Segunda Geração: linguagens de montagem (assembly).
- Terceira Geração: linguagens procedurais.
- Quarta Geração: linguagens aplicativas.
- Quinta Geração: linguagens voltadas a Inteligência artificial como as linguagens lógicas (Prolog) e as linguagens funcionais (Lisp).
- Sexta Geração: redes neurais.

Evolução

Introdução

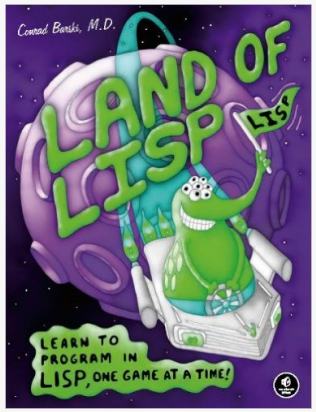
Já tivemos um resumo histórico sobre as linguagens de programação. Veremos agora alguns aspectos sobre a evolução linguagens, sem nos aprofundar muito no tema, apenas para saber como as coisas chegaram onde estão hoje em dia.

Ao lado temos um infográfico disponível no site siliconangle.com.

Clique na imagem ao lado para acessar o site e observar a imagem em alta definição.

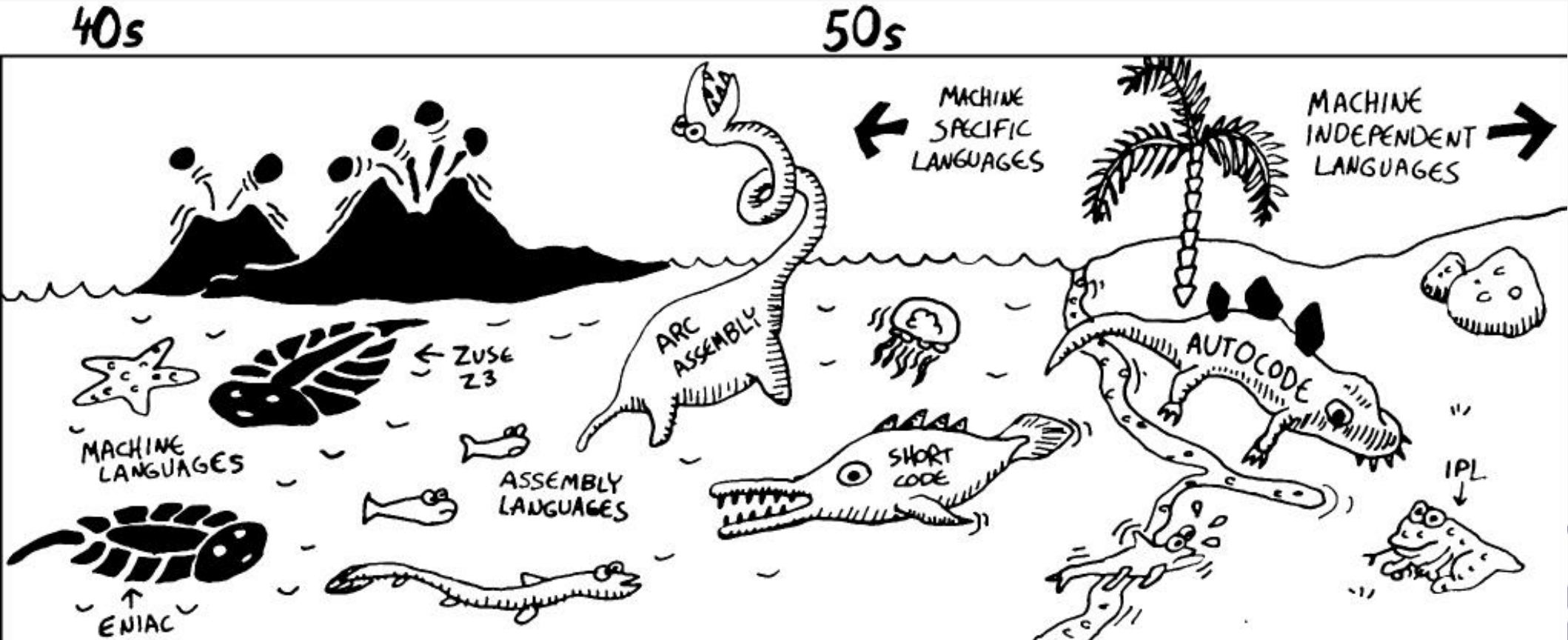
Evolução

Introdução



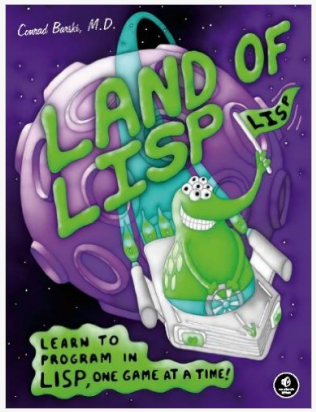
O livro "Land of Lisp: Learn to Program in Lisp, One Game at a Time!", do autor Conrad Barski, traz uma série de figuras divertidas sobre a evolução das linguagens de programação.

O autor foca na linguagem LISP. Mas é divertido observar as imagens, analisando o surgimento e evolução de algumas linguagens.



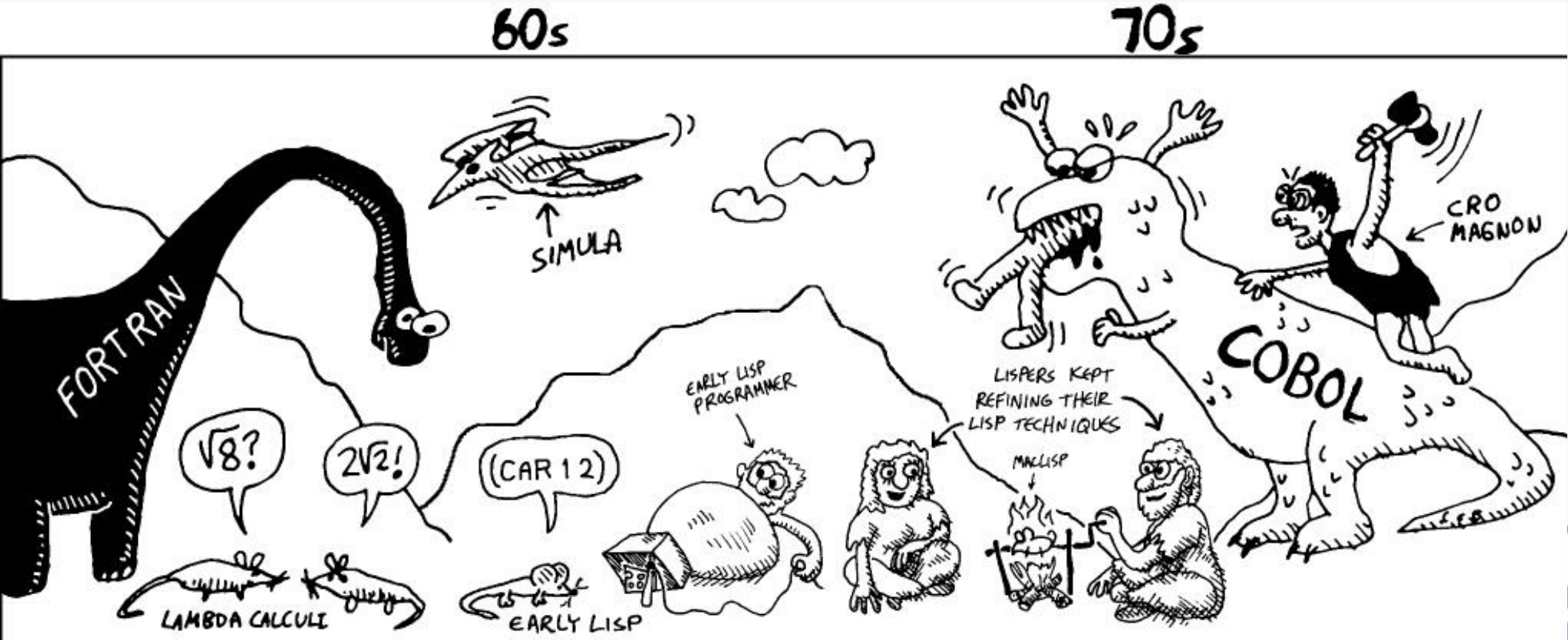
Evolução

Introdução



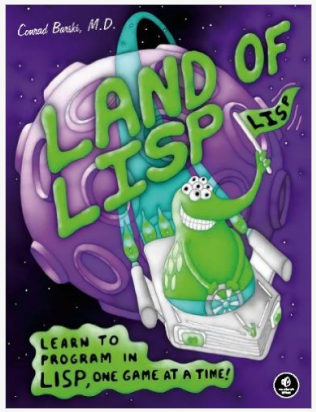
O livro “Land of Lisp: Learn to Program in Lisp, One Game at a Time!”, do autor Conrad Barski, traz uma série de figuras divertidas sobre a evolução das linguagens de programação.

O autor foca na linguagem LISP. Mas é divertido observar as imagens, analisando o surgimento e evolução de algumas linguagens.



Evolução

Introdução

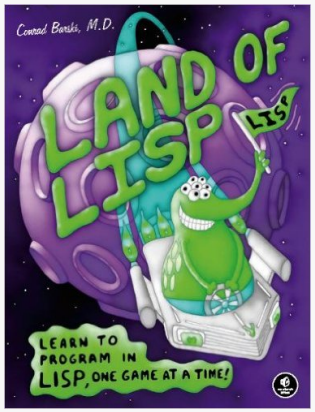


O livro "Land of Lisp: Learn to Program in Lisp, One Game at a Time!", do autor Conrad Barski, traz uma série de figuras divertidas sobre a evolução das linguagens de programação.

O autor foca na linguagem LISP. Mas é divertido observar as imagens, analisando o surgimento e evolução de algumas linguagens.

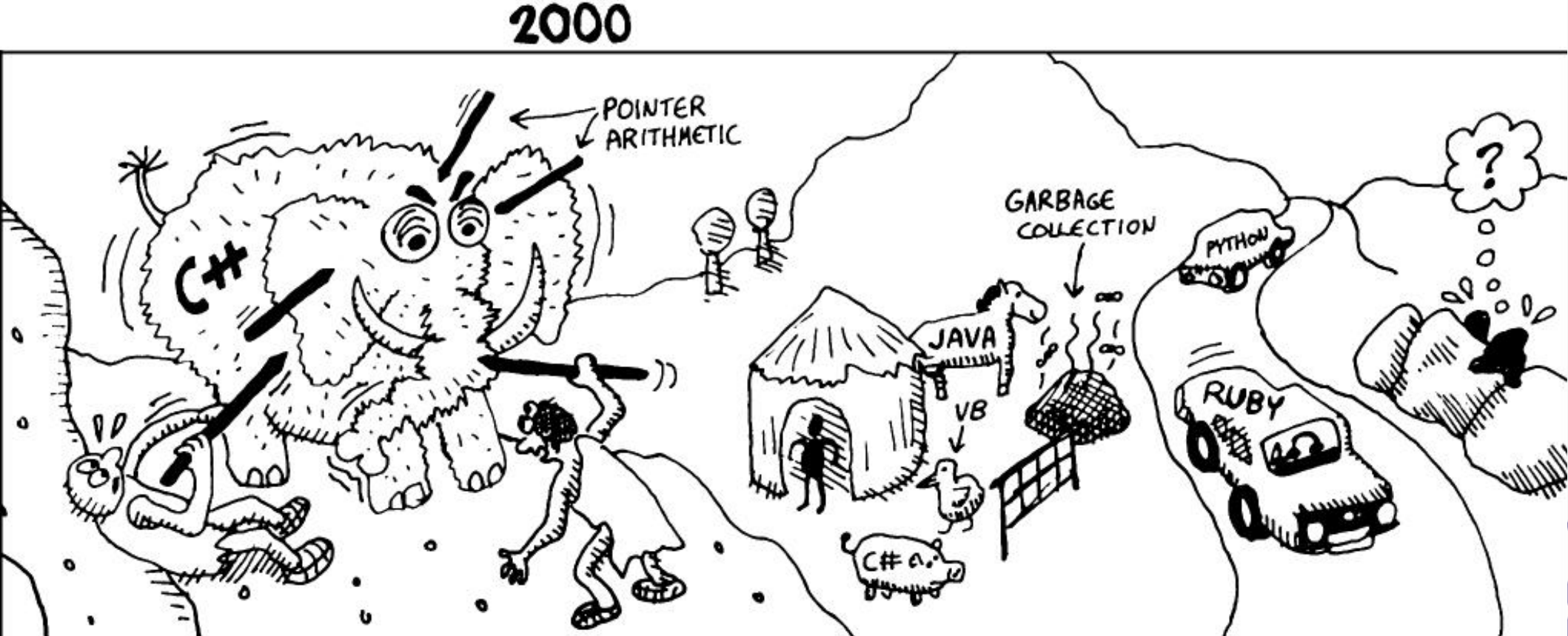
Evolução

Introdução



O livro "Land of Lisp: Learn to Program in Lisp, One Game at a Time!", do autor Conrad Barski, traz uma série de figuras divertidas sobre a evolução das linguagens de programação.

O autor foca na linguagem LISP. Mas é divertido observar as imagens, analisando o surgimento e evolução de algumas linguagens.



Evolução

Genealogia

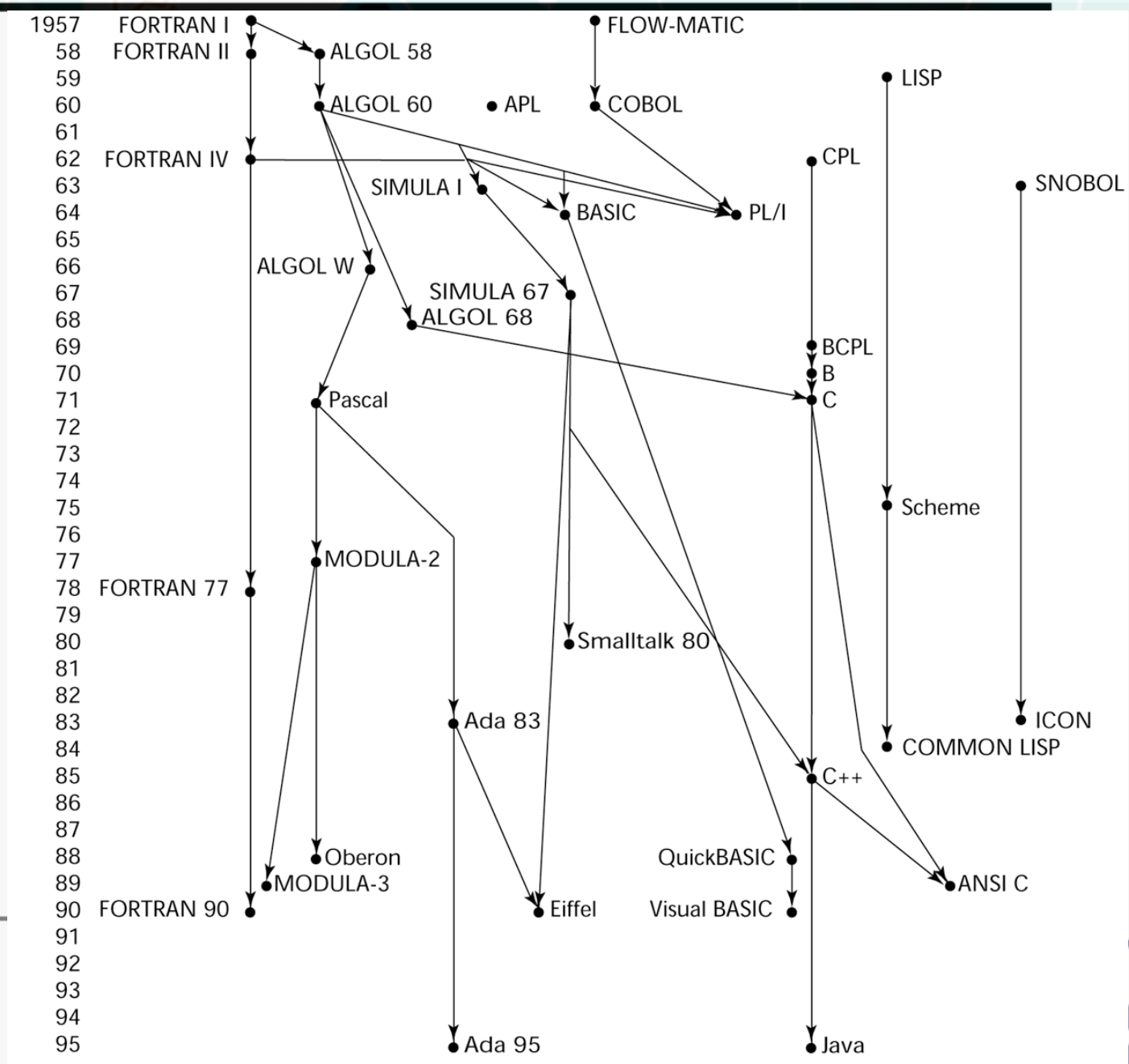
SEBESTA (2011) nos apresenta a genealogia das linguagens de programação.

Observe o Java, por exemplo. Essa linguagem tem aspectos do C++, C, ALGOL e Fortran.

Esse gráfico é bem fraquinho. Dá pra ter uma noção, mas ele traz apenas algumas linguagens. Convido o leitor a clicar na imagem ao lado para ser direcionado para o site

https://www.levenez.com/lang/

Clique na opção "Plotter" e veja uma genealogia bem mais completa, de 1954 a 2015. Divirta-se!



T2Ti.COM

Evolução

Antes de 1940

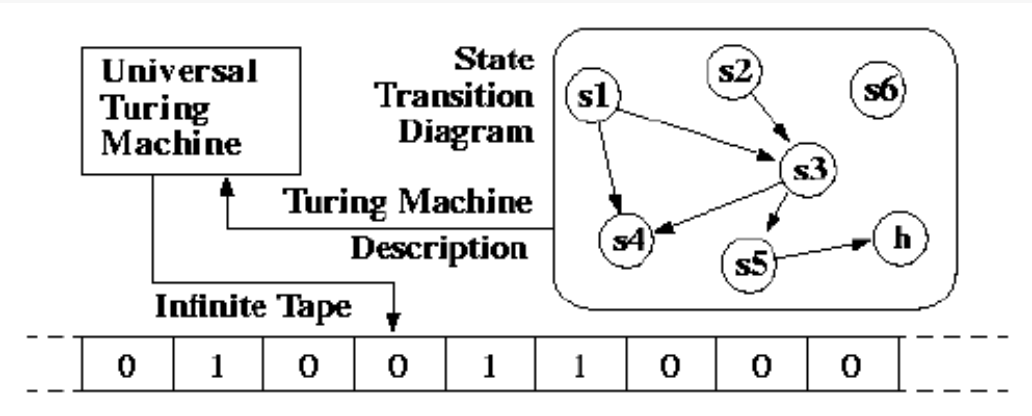
As linguagens de programação são anteriores ao advento do primeiro computador moderno. De início as linguagens eram apenas códigos.

Durante um período de nove meses entre 1842-1843, Ada Lovelace traduziu as memórias do matemático italiano Luigi Menabrea sobre a a mais nova máquina proposta por Charles Babbage, a sua máquina analítica. Com o artigo, ela anexou uma série de anotações que especificavam em completo detalhe um método para calcular números de Bernoulli com a máquina, reconhecido por alguns historiadores como o primeiro programa de computador do mundo.

Herman Hollerith percebeu que poderia codificar a informação em cartões perfurados quando ele observou que o condutor de trens controlava a presença dos titulares dos bilhetes com a posição dos furos no bilhete. Hollerith, então, começou a codificar os dados do censo de 1890 em cartões perfurados.

A primeira linguagem de programação moderna é difícil de ser identificada. No início, as restrições do hardware definiam a linguagem. Cartões perfurados dispunham de até 80 colunas, mas algumas das colunas tinham que ser usados para um número de sequência de cada cartão.

Fortran incluía algumas palavras-chave que eram as mesmas palavras em inglês, como "IF" (se), "GOTO" (vá para) e "CONTINUE" (continue). O uso de um tambor magnético para a memória significava que os programas de computador também tinham que ser intercalados com as rotações do tambor. Assim, os programas eram mais dependentes do hardware do que hoje.



Evolução

A Década de 1940

Na década de 1940 os primeiros computadores elétricos, reconhecidamente modernos, foram criados.

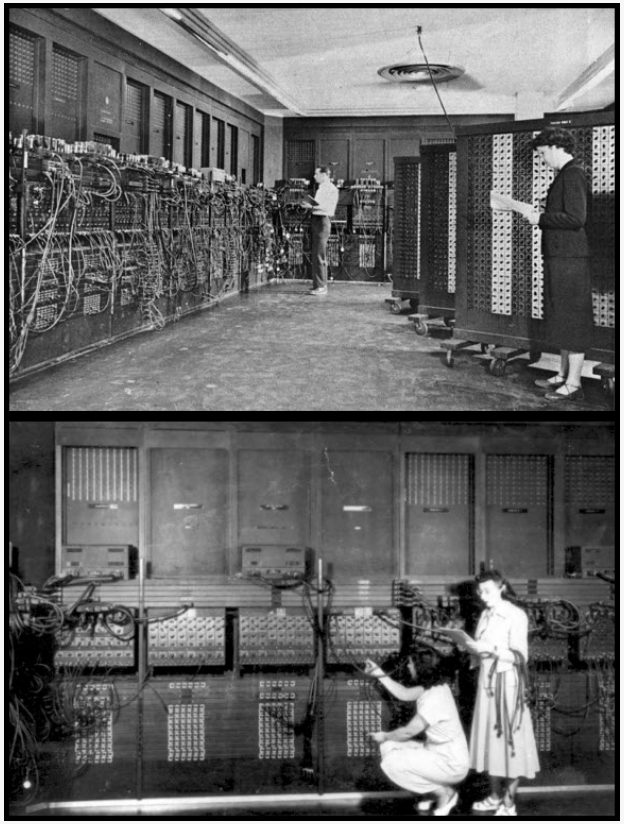
A limitada capacidade da memória forçava os programadores a escrever à mão economicamente programas em linguagem de montagem (linguagem de máquina).

Logo se descobriu que a programação em linguagem assembly exigia um grande esforço intelectual e era muito sujeita a erros.

Em 1948, Konrad Zuse publicou um artigo sobre a sua linguagem de programação Plankalkül. No entanto, esta não foi implementada em sua época e suas contribuições originais foram isoladas de outros desenvolvimentos.

Algumas linguagens importantes que foram desenvolvidas durante este período incluem:

- 1943 - Plankalkül (Konrad Zuse)
- 1943 - ENIAC coding system
- 1949 – C-10



Evolução

As Décadas de 1950 e 1960

Na década de 1950 as primeiras três linguagens de programação modernas, cujos descendentes ainda estão em uso difundido hoje foram concebidas:

- FORTRAN (1954), a "FORmula TRANslator", inventada por John Backus e outros.
- LISP, a "LISt Processor", inventada por John McCarthy e outros.
- COBOL, a COmmon Business Oriented Language, criada pelo Short Range Committee, com grande influência de Grace Hopper.

Outro marco na década de 1950 foi a publicação, por um comitê de cientistas americanos e europeus, de "uma nova linguagem para os algoritmos", a ALGOL 60 através da publicação do relatório "The ALGOL 60 Report". Este relatório consolidou muitas ideias que circulavam na época e apresentou duas inovações chave quanto ao projeto de linguagens:

- Estrutura de blocos aninhados: pedaços significativos de código poderiam ser agrupados em bloco de instruções, sem ter que ser transformados em procedimentos separados e ser explicitamente chamados.
- Escopo léxico: um bloco podia ter suas próprias variáveis não acessíveis fora do bloco, e muito menos manipuláveis de fora do bloco.

Outra inovação, relacionada a esta última, foi na forma como a linguagem foi descrita:

- Uma notação matemática exata, Backus-Naur (BNF - é uma meta sintaxe usada para expressar gramáticas livres de contexto, isto é, um modo formal de descrever linguagens formais.), foi utilizada para descrever a sintaxe da linguagem. Quase todas as linguagens de programação posteriores utilizaram uma variante da BNF para descrever a parte livre de contexto de sua sintaxe.

Evolução

As Décadas de 1950 e 1960

Algol 60 foi particularmente influente na concepção das linguagens posteriores, algumas das quais logo se tornaram mais populares.

Algumas ideias-chave da linguagem Algol foram tomadas, produzindo-se a linguagem ALGOL 68:

- A sintaxe e semântica se tornaram ainda mais ortogonais, com rotinas anônimas, um sistema recursivo de digitação com funções de ordem superior, etc.
- Não somente a parte livre de contexto da linguagem, mas a sintaxe da linguagem completa e a semântica foram definidos formalmente, em termos da gramática de Van Wijngaarden, um formalismo desenvolvido especificamente para esta finalidade.

Os recursos pouco utilizados de Algol 68 (por exemplo, blocos simultâneos e paralelos) e seu complexo sistema de atalhos sintáticos e coerções de tipo automático tornou a linguagem impopular entre os implementadores e ganhou a reputação de ser difícil. Niklaus Wirth realmente saiu do comitê de projeto para criar uma linguagem mais simples: Pascal.

Panorama das duas décadas:

- 1954 - FORTRAN
- 1955 - FLOW-MATIC (antecessor do COBOL)
- 1957 - COMTRAN (antecessor do COBOL)
- 1958 - LISP
- 1958 - ALGOL 58
- 1959 - FACT (antecessor do COBOL)
- 1959 - COBOL
- 1962 - APL
- 1962 - Simula
- 1964 - BASIC
- 1964 - PL/I

1967 a 1978 | Paradigmas Fundamentais

A maioria dos principais paradigmas de linguagem agora em uso foram inventados durante este período:

- Simula: inventada nos anos 1960 por Nygaard e Dahl como um super conjunto de Algol 60, foi a primeira linguagem a suportar o conceito de classes.
- C: uma das primeiras linguagens de programação de sistemas, foi desenvolvido por Dennis Ritchie e Ken Thompson nos laboratórios da Bell entre 1969 e 1973.
- Smalltalk: feito em meados de 1970, forneceu uma base completa para o projeto de uma linguagem orientada a objetos.
- Prolog: projetada em 1972 por Colmerauer, Roussel, e Kowalski, foi a primeira linguagem de programação do paradigma lógico.
- ML: inventada por Robin Milner em 1973, uma linguagem funcional, baseada em Lisp, estaticamente tipada.

Cada uma dessas linguagens gerou toda uma família de descendentes. As linguagens mais modernas contam, pelo menos, com uma delas em sua ascendência.

As décadas de 1960 e 1970 também viram um considerável debate sobre os méritos da "programação estruturada", que, entre outros aspectos, envolvia a programação sem o uso de GOTO. Apesar de o debate ter ficado quente na época, quase todos os programadores agora concordam que, mesmo em linguagens que fornecem um comando de desvio incondicional como o GOTO, é um mau estilo de programação usá-lo, exceto em raras circunstâncias.

Algumas linguagens importantes que foram desenvolvidas durante este período incluem: Pascal (1970), Forth (1970), C (1972), Smalltalk (1972), Prolog (1972), ML (1973), SQL (1978).

Evolução

Anos 80 | Consolidação, Módulos, Performance

Os anos 1980 foram de relativa consolidação. O C++ combinou a orientação a objetos com a programação de sistemas. O governo dos Estados Unidos padronizou a Ada, uma linguagem de programação de sistemas destinados à utilização por parte dos contratantes de defesa.

No Japão e em outros lugares, vastas somas foram gastas investigando as chamadas linguagens de programação de quinta geração que incorporavam a programação lógica em suas construções.

A comunidade de linguagens funcionais se dedicou a padronizar a ML e o Lisp. No lugar de inventar novos paradigmas, todos estes esforços visaram aperfeiçoar as ideias inventadas na década anterior.

Embora os principais paradigmas novos para as linguagens de programação não tivessem aparecido, muitos pesquisadores expandiram as ideias das linguagens existentes adaptando-os para novos contextos. Por exemplo, as linguagens dos sistemas Argus e Emerald adaptaram a programação orientada a objeto para os seus sistemas distribuídos.

Algumas linguagens importantes que foram desenvolvidas durante este período incluem:

- 1983 - Ada
- 1983 - C++
- 1985 - Eiffel
- 1987 - Perl
- 1989 - FL (Backus)

Evolução

Anos 90 | A Era da Internet

A década de 1990 não viu nenhuma novidade fundamental, mas a recombinação e maturação das ideias antigas.

Uma filosofia de grande importância era a produtividade do programador. Muitas linguagens com RAD (aplicações de desenvolvimento rápido) surgiram, geralmente vindo com uma IDE, coleta de lixo, e eram descendentes de linguagens mais antigas.

Todas essas linguagens foram orientadas a objeto. Entre estas estavam a Object Pascal, Visual Basic, e C#. Java era uma linguagem mais conservadora, que também incluiu a coleta de lixo e recebeu muita atenção.

Mais radicais e inovadoras do que as linguagens RAD foram as novas linguagens de script. Estas não descenderam diretamente das outras linguagens e contaram com sintaxes novas e incorporação mais liberal de novas funcionalidades.

Muitos consideram essas linguagens de script mais produtivas do que até mesmo as linguagens RAD, por causa da facilidade com que pequenos programas podem ser escritos e mantidos. No entanto, linguagens de script vieram a ser mais utilizadas em conexão com a web.

Algumas linguagens importantes que foram desenvolvidas durante este período incluem:

- 1990 - Haskell
- 1991 - Python
- 1991 - Java
- 1993 - Ruby
- 1993 - Lua
- 1994 - ANSI Common Lisp
- 1995 - JavaScript
- 1995 - PHP
- 2000 - C#
- 2008 - JavaFX Script
- 2011 – Dart
- 2012 - TypeScript

Evolução

Tendências

A evolução das linguagens de programação continua, tanto na indústria quanto na pesquisa. Algumas das tendências incluem:

- Mecanismos para a adição de segurança e verificação da confiabilidade para a linguagem: verificação estática prolongada, controle de fluxo de informação, estático segurança em threads.
- Mecanismos alternativos de modularidade.
- Programação orientada a aspectos.
- Desenvolvimento de software orientado a componentes.
- Metaprogramação, Reflexão ou acesso a árvores de sintaxe abstratas.
- Maior ênfase na distribuição e mobilidade.
- Integração com bases de dados, incluindo XML e bancos de dados relacionais.
- Suporte para Unicode de forma que o código-fonte não esteja restrito aos caracteres contidos no código ASCII.
- XML para a interfaces gráficas (XUL, XAML).

Tópicos Importantes

Variáveis

Na programação, uma variável é um objeto (uma posição, frequentemente localizada na memória) capaz de reter e representar um valor ou expressão. Enquanto as variáveis só "existem" em tempo de execução, elas são associadas a "nomes", chamados identificadores, durante o tempo de desenvolvimento.

Quando nos referimos à variável, do ponto de vista da programação de computadores, estamos tratando de uma "região de memória (do computador) previamente identificada cuja finalidade é armazenar os dados ou informações de um programa por um determinado espaço de tempo".

A memória do computador se organiza tal qual um armário com várias divisões. Sendo cada divisão identificada por um endereço diferente em uma linguagem que o computador entende.

O computador armazena os dados nessas divisões, sendo que em cada divisão só é possível armazenar um dado e toda vez que o computador armazenar um dado em uma dessas divisões, o dado que antes estava armazenado é eliminado. O conteúdo pode ser alterado, mas somente um dado por vez pode ser armazenado naquela divisão.

O computador identifica cada divisão por intermédio de um endereço no formato hexadecimal, e as linguagens de programação permitem nomear cada endereço ou posição de memória, facilitando a referência ao endereço de memória.

Uma variável é composta por dois elementos básicos: o conteúdo (o valor da variável) e o identificador (um nome dado à variável para possibilitar sua utilização).

Tipos de Dados

O que seriam os tipos de dados? Vimos na página anterior que podemos criar variáveis para ter acesso a valores que estão na memória. Por exemplo:

A = "albert";
B = 125;

Temos duas variáveis, cujos identificadores são "A" e "B". Mas observe que a primeira contém um texto e a segunda contém um número. É aí que entram os tipos de dados.

Um tipo de dado define uma coleção de objetos de dados e um conjunto de operações pré-definidas sobre estes objetos.

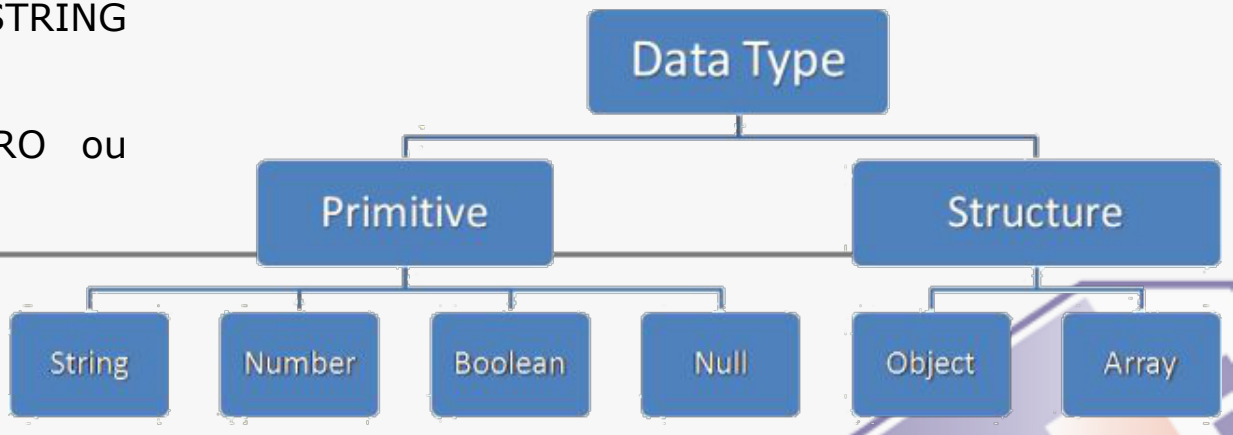
A variável "A" seria do tipo TEXTO ou STRING ou VARCHAR, etc.

A variável "B" seria do tipo INTEIRO ou INTEGER ou NUMBER, etc.

Os tipos de dados são uma combinação de valores e de operações que uma variável pode executar, o que pode variar conforme o sistema operacional e a linguagem de programação.

São utilizados para indicar ao compilador ou interpretador as conversões necessárias para obter os valores em memória durante a construção do programa. O tipo de dado ajuda também o programador a detectar eventuais erros envolvidos com a semântica das instruções.

Os tipos têm geralmente associações com valores na memória ou com objetos (para uma linguagem orientada a objeto) ou variáveis.



Tópicos Importantes

Tipos de Dados

Como já mencionado, cada linguagem definirá os seus tipos de dados.

Podemos definir algumas classificações, a saber:

- Tipos primitivos e compostos.
- Tipos fortes e fracos.
- Tipos estáticos e dinâmicos.

Tipos Primitivos e Compostos

Um tipo primitivo (também conhecido por nativo ou básico) é fornecido por uma linguagem de programação como um bloco de construção básico.

Um tipo composto pode ser construído em uma linguagem de programação a partir de tipos primitivos e de outros tipos compostos, em um processo chamado composição.

Tipos primitivos típicos incluem caractere, inteiro (representa um subconjunto dos números inteiros, com largura dependente do sistema e pode possuir sinal ou não), ponto flutuante (representa o conjunto dos números reais), booleano (lógica booleana, verdadeiro ou falso) e algum tipo de referência (como ponteiro ou handles).

Tipos primitivos mais sofisticados incluem tuplas, listas ligadas, números complexos, números racionais e tabela hash, presente sobretudo em linguagens funcionais.

Espera-se que operações envolvendo tipos primitivos sejam as construções mais rápidas da linguagem. Por exemplo, a adição de inteiros pode ser feita com somente uma instrução de máquina.

Os tipos compostos se dividem em homogêneos (vetores e matrizes) e heterogêneos (registros).

Tópicos Importantes

Tipos de Dados

Em algumas universidades existe uma disciplina específica só para estudar os tipos compostos, muitas vezes chamada de Estrutura de Dados.

Como já mencionado, os tipos compostos se dividem em homogêneos (vetores e matrizes) e heterogêneos (registros).

- As estruturas homogêneas são conjuntos de dados formados pelo mesmo tipo de dado primitivo.
- As estruturas heterogêneas são conjuntos de dados formados por tipos de dados primitivos diferentes (campos do registro) em uma mesma estrutura.

O estudo das estruturas de dados está em constante desenvolvimento (assim como o de algoritmos), mas, apesar disso, existem certas estruturas clássicas que se comportam como padrões.

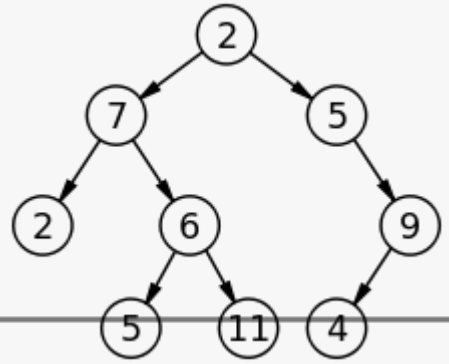
- Vetores ou arrays: são estruturas de dados lineares e estáticas, isto é, são compostas por um número fixo (finito) de elementos de um determinado tipo de dados. O tempo de acesso aos elementos de um vetor é muito rápido, sendo considerado constante: o acesso aos elementos é feito pelo seu índice no vetor. Porém, a remoção de elementos pode ser custosa se não for desejável que haja espaços "vazios" no meio do vetor, pois nesse caso é necessário "arrastar" de uma posição todos os elementos depois do elemento removido. Essa é uma estrutura muito recomendada para casos em que os dados armazenados não mudarão, ou pouco mudarão, através do tempo.
- Lista: é uma estrutura de dados linear. Uma lista ligada, também chamada de encadeada, é composta por nós que apontam para o próximo elemento da lista, o último elemento apontará para nulo. Para compor uma lista encadeada, basta guardar seu primeiro elemento.

Tópicos Importantes

Tipos de Dados

- Fila: são estruturas baseadas no princípio FIFO (first in, first out), em que os elementos que foram inseridos no início são os primeiros a serem removidos. Uma fila possui duas funções básicas: ENQUEUE, que adiciona um elemento ao final da fila, e DEQUEUE, que remove o elemento no início da fila. A operação DEQUEUE só pode ser aplicada se a fila não estiver vazia, causando um erro de underflow ou fila vazia se esta operação for realizada nesta situação.
- Pilha: é uma estrutura de dados baseada no princípio LIFO (LAST in, FIRST out), na qual os dados que foram inseridos primeiros na pilha serão os últimos a serem removidos. Existem duas funções que se aplicam a todas as pilhas: PUSH, que insere um dado no topo da pilha, e POP, que remove o item no topo da pilha.
- Árvore: é uma estrutura de dados em que cada elemento tem um ou mais elementos associados, podendo definir-se uma árvore recursivamente como: (1) uma estrutura (uma árvore); (2) um nó (designado por raiz), que contém a informação a armazenar e um conjunto finito de árvores (as subárvores).
- Árvore Binária: é uma árvore em que cada nó tem no máximo dois filhos. São muito utilizadas como estruturas de buscas.

Além dessas estruturas, podemos citar o Grafo, o Deque e a Tabela de Hash (hash table ou hash map).



Tipos de Dados

Tipos Fortes e Fracos

Linguagens implementadas com tipificação forte (linguagem fortemente tipificada), tais como Java e Pascal, exigem que o tipo de dado de um valor seja do mesmo tipo da variável ao qual este valor será atribuído. Exemplo:

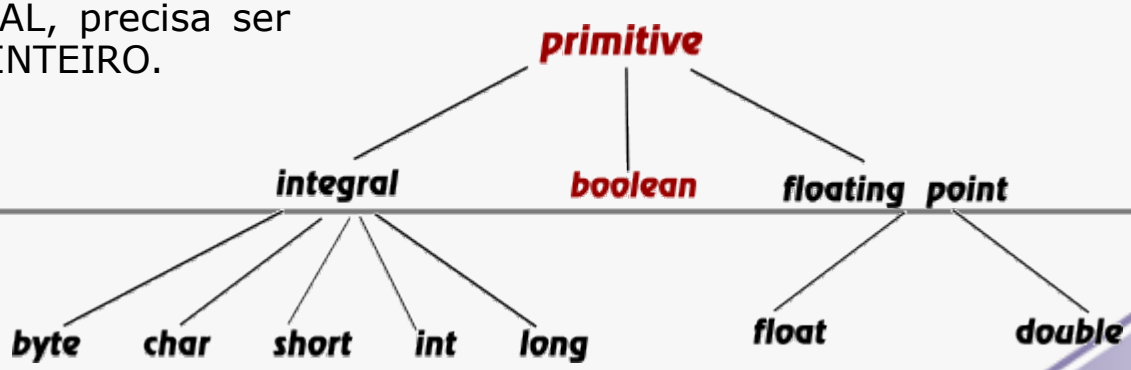
```
1. Declaração de Variáveis
2.  TEXTO nome
3.  INTEIRO idade
4.
5. Atribuições
6.  nome = "Albert"
7.  idade = 25.2
```

Ocorrerá um erro ao compilar a linha 7, pois o valor "25.2", que é do tipo REAL, precisa ser convertido para o tipo de dado INTEIRO.

Em linguagens com tipos de dados fracos, tais como PHP e VBScript, a conversão não se faz necessária, sendo realizada implicitamente pelo compilador ou interpretador. Se o programa anterior fosse escrito em PHP, não ocorreria nenhum erro. Uma vantagem da tipificação forte é o fato de que a mesma permite a detecção do mau uso de varáveis que resultam em erros de tipo.

Tipos Estáticos e Dinâmicos

A verificação do tipo de dado é feita de forma estática em tempo de compilação ou de forma dinâmica em tempo de execução. Em C, C++, Java e Haskell os tipos são estáticos, em Scheme, Lisp, Smalltalk, Perl, PHP, Visual Basic, Ruby e Python são dinâmicos.



Tópicos Importantes

Estruturas de Controle

Em ciência da computação, estrutura de controle (ou fluxo de controle) refere-se à ordem em que instruções, expressões e chamadas de função são executadas ou avaliadas em programas de computador sob programação imperativa ou funcional.

Para ressumir: são os For, While, Ifs e Cases da vida.

Os tipos de estruturas de controle disponíveis diferem de linguagem para linguagem, mas podem ser caracterizados por seus efeitos.

Nas primeiras linguagens era muito comum o uso do GOTO, que fazia um desvio incondicional do código. Era um mau estilo de programação, visto que o tal desvio era motivo para vários erros de programação.

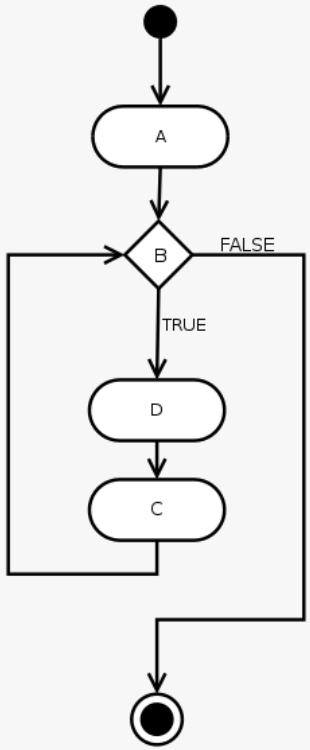
Podemos resumir as estruturas em três tipos: sequenciais, de seleção e de repetição.

A estrutura sequencial realiza um conjunto predeterminado de comandos de forma sequencial, de cima para baixo, na ordem em que foram declarados.

A estrutura de seleção realiza diferentes ações dependendo se a seleção (ou condição) é verdadeira ou falsa, em que a expressão é processada e transformada em um valor booleano. Exemplos: If...Then...Else; Case...Of.

A estrutura de repetição realiza e/ou repete diferentes algoritmos/ações dependendo se uma condição é verdadeira ou falsa. Exemplos: While...Do; For...; Repeat...Until.

for(A;B;C)
D;



Tópicos Importantes

Subprograma ou Sub-rotina

Uma sub-rotina (função, procedimento ou mesmo subprograma) consiste em uma porção de código que resolve um problema muito específico, parte de um problema maior (a aplicação final).

O conceito de função difere da noção de procedimento, já que devolve um valor, se bem que, em algumas linguagens, esta distinção não existe. Por exemplo, em C, a implementação de um procedimento é uma função do tipo void (sem retorno).

No contexto da programação orientada a objetos, estas sub-rotinas são encapsuladas nos próprios objetos, passando a designar-se métodos.

Quais as vantagens de usar sub-rotinas? Seguem algumas:

- Redução de código duplicado num programa.
- Possibilidade de reutilizar o mesmo código sem grandes alterações em outros programas.
- Decomposição de problemas grandes em pequenas partes.
- Melhorar a interpretação visual de um programa.
- Esconder ou regular uma parte de um programa, mantendo o restante do código alheio às questões internas resolvidas dentro dessa função.

Componentes da sub-rotina:

- Seu protótipo (assinatura), que inclui os parâmetros que são passados no momento da invocação.
- O corpo, que contém o bloco de código que resolve o problema proposto.
- Um possível valor de retorno.

Tópicos Importantes

Passagem de Parâmetros

Os identificadores utilizados na assinatura de um método, ou seja, na definição de uma sub-rotina são chamados de parâmetros formais, ou simplesmente parâmetros.

Os identificadores que são utilizados na chamada aos subprogramas são chamados de parâmetros reais, ou simplesmente argumentos.

Há 3 formas distintas de se fazer a passagem de parâmetros para um subprograma: passagem por referência, passagem por cópia e passagem por nome. A maioria das linguagens usa o critério posicional para fazer a amarração entre argumentos e parâmetros.

Exemplo:

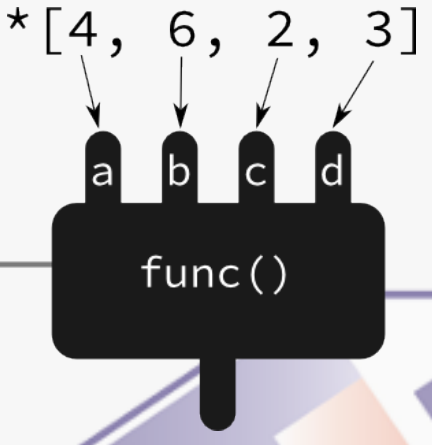
```
procedure Teste (p1, p2, ..., pn);

Chamada:
Teste (a1, a2, ..., an);
```

Nesse caso, "p1, p2, ... , pn" são os parâmetros e "a1, a2, ..., an" são os argumentos. Pelo critério posicional, o primeiro argumento será passado para o primeiro parâmetro (p1 := a1), o segundo argumento será passado para o segundo parâmetro (p2 := a2), e assim por diante.

Passagem por Referência

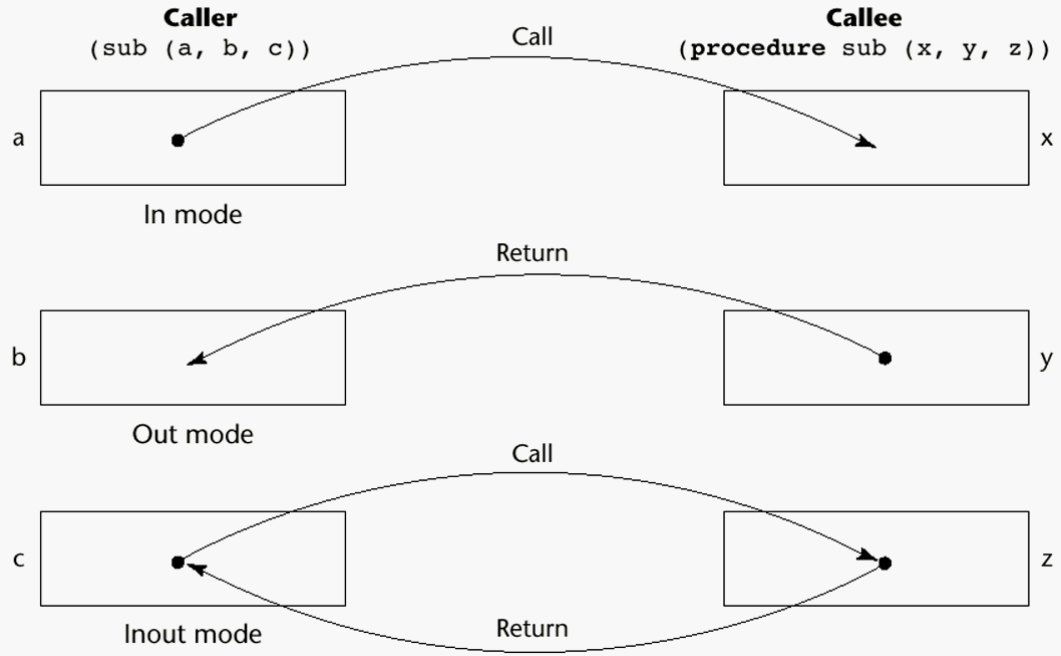
A função chamadora passa para a função chamada apenas o endereço do argumento. Dessa forma, a variável usada como argumento é compartilhada entre as duas funções e eventuais alterações dentro da função chamada serão refletidas diretamente no argumento.



Passagem de Parâmetros

Passagem por Cópia

Os parâmetros se comportam como variáveis locais. Podemos subdividir a passagem por cópia em três tipos: passagem por valor (in mode), passagem por resultado (out mode), e passagem por valor-resultado (inout mode).



Por valor: os argumentos são usados para inicializar os parâmetros, que funcionam como variáveis locais. Os parâmetros são usados para receber os valores trazidos pelos argumentos. Não retornam valores através dos parâmetros. Se os parâmetros forem do tipo ponteiro, a passagem por valor se comportará igual à passagem por referência.

Por Resultado: Diferentemente da passagem por valor, os parâmetros não recebem valor quando o subprograma é chamado. Assim como na passagem por valor, os parâmetros também funcionam como variáveis locais. Esse mecanismo é semelhante àquele utilizado por uma função para devolver o resultado do seu processamento – a diferença consiste apenas na notação: a função usa o seu próprio nome para retornar com o resultado, enquanto que este mecanismo usa um parâmetro ou uma palavra-chave dentro da função, como *result* ou *return*.

Índice Tiobe

Apresentação

O Índice TIOBE (do inglês, TIOBE Programming Community Index) é uma lista ordenada de linguagens de programação, classificada pela frequência de pesquisa na web usando o nome da linguagem como a palavra-chave. O índice é atualizado mensalmente. De acordo com o site, o índice TIOBE não é sobre a melhor linguagem de programação, ou em qual se tem escrito a maior quantidade de linhas de código. Entretanto, o site alega que a frequência de buscas pode refletir o número de engenheiros hábeis, cursos e vagas de emprego no mundo todo. Clique na imagem para acessar o índice.

Jan 2016	Jan 2015	Change	Programming Language	Ratings	Change
1	2	▲	Java	21.465%	+5.94%
2	1	▼	C	16.036%	-0.67%
3	4	▲	C++	6.914%	+0.21%
4	5	▲	C#	4.707%	-0.34%
5	8	▲	Python	3.854%	+1.24%
6	6		PHP	2.706%	-1.08%
7	16	⇈	Visual Basic .NET	2.582%	+1.51%

Referências

ABREU, M. C. O Professor Universitário em Sala de Aula. São Paulo, MG Associados, 1990

BERRY, D. M.; LAWRENCE, B. Requirements engineering. 1. ed.

BEZERRA, E. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro, Campus, 2002.

FIORINI, S. T. & STAA, A. & BAPTISTA, R. M. Engenharia de Software com CMM. Rio de Janeiro, Brasport, 1998.

GAUSE, D. C.; WEINBERG, G. M. Are your lights on? 1. ed. USA: Dorset House Publishing Co. Inc., 1990.

MULLER, R. J. Projeto de Banco de Dados. São Paulo, Berkeley, 2002.

REZENDE, D. A. Engenharia de Software e Sistemas de Informação. Rio de Janeiro, Brasport, 2002.

SOMMERVILLE, I. Engenharia de Software. São Paulo, 2011.

SEBESTA, R. Conceitos de linguagens de programação. 9. ed. Porto Alegre: Bookman, 2011.

MACLENNAN, Bruce J. Principles of Programming Languages: Design, Evaluation and Implementation (em inglês). 3ª ed. Oxford: Oxford University Press, 1999.

BAL, Henri E.; Grune, Dick. Programming Language Essentials (em inglês). Wokingham: Addison-Wesley, 1994.

Wikipedia, Consultas em Janeiro/2016.