Primeira geração de linguagens de programação (máquina)

A primeira geração de linguagens remonta aos dias da codificação em linguagem de máquina, surgidas com o início da computação na década de 50, especificamente de 1950 a 1962. A Linguagem de máquina e Assembly representam esta primeira geração das linguagens de programação. Essas linguagens totalmente dependentes da máquina, exibem o mais baixo nível de abstração que uma linguagem pode ser representada. Elas somente devem ser usadas quando as linguagens de mais alto nível não satisfizerem as necessidades ou não forem suportadas. **(PIMENTEL, 2014)**

Segunda geração das linguagens de programação (novidades)

A segunda geração de linguagens de programação foi desenvolvida de 1962 a 1974 e serviu de base para o desenvolvimento das modernas linguagens de programação. As características marcantes das linguagens de segunda geração foram o amplo uso com grande familiaridade e aceitação no mercado, sistemas de execução em tempo real e desenvolvimento de gerenciadores de base de dados. As linguagens Fortran, Cobol, Algol e algumas extensões como Basic, foram os representantes dessa segunda geração. **(PIMENTEL, 2014)**

ALGOL *(Algorithm lenguage)*

ALGOL foi o precursor de muitas linguagens da terceira geração, por oferecer ricamente estruturas de controle e tipos de dados.

O ALGOL 60 foi produzido por um grupo de pessoas, metade indicado pela ACM e metade por várias instituições europeias. Além desse patrocínio inicial, o grupo não possuía posição oficial ou autoridade. Entretanto, a Federação Internacional de Processamento de Informação (IFIP), (fundada em 1960), através dos auspícios de seu Comitê Técnico 2 sobre Linguagens de Programação (TC2), fundou um grupo de trabalho sobre o ALGOL. Em sua reunião em Roma, em abril de 1962, concluiu-se do Relatório ALGOL 60 Revisto, e os autores aceitaram que qualquer responsabilidade coletiva que possam ter em relação ao desenvolvimento, especificação e refinamento da linguagem ALGOL será, a partir de agora, transferida. (BERGIN e GIBSON, 1996 - adaptado)

A maioria dos membros do Grupo de projeto do ALGOL 68 estavam lá por conta de seu envolvimento ou experiência com ALGOL 60. Aqueles pertencentes a instituições acadêmicas teriam deveres de ensino e pesquisa/interesses não necessariamente ligados diretamente com ALGOL 68. Apenas a equipe em Amsterdã (VanWijngaarden, Mailloux, Peck e Koster) juntamente com Goos, que estava preparando uma implementação,trabalhou em tempo integral no projeto. (BERGIN e GIBSON, 1996 - adaptado)

```
let n = readi
let x = vector 1::n of 0.0
for i = 1 to n do x( i ) := readr
!bubble sort starts here
for i = 1 to n - 1 do
   for j = 1 to n - i do
   if x( j ) > x( j + 1 ) do
   begin
        let temp = x( j )
        x( j ) := x( j + 1 )
        x( j + 1 ) := temp
   end
!now write out the answers
write "The sorted numbers are'n'n"
for i = 1 to n do write x( i ),"'n" ?
```

trecho de código em ALGOL

(Disponível em: <https://craftofcoding.wordpress.com/2017/06/16/ghosts-in-the-machine-algol-60/>)

BASIC *(Beginner's All-purpose Symbolic Instruction Code)*

Basic foi uma linguagem originalmente criada para o aprendizado e teve seu uso bastante reduzido já na década de 70. (PIMENTEL, 2014)

Em 1964, os alunos do Dartmouth College precisavam de melhor acesso a computadores e de uma linguagem simples e eficaz para escrever programas de computador. Primeiro, John G. Kemeny, que era presidente do Departamento de Matemática, e o professor Thomas Kurtz criaram o Dartmouth Time-Sharing System, um dos primeiros sistemas de computadores compartilhados no tempo nos Estados Unidos; depois, criaram a linguagem de programação BASIC para que os alunos pudessem escrever programas para serem executados no mainframe General Electric GE-225, que era o coração do sistema. Em 1º de maio de 1964, dois programas escritos no BASIC foram executados simultaneamente no sistema compartilhado pelo tempo de Dartmouth e ambas as realizações mudariam a computação para sempre. Dez anos depois, uma versão do BASIC escrita por Bill Gates e Paul Allen foi incluída no Altair, o primeiro computador pessoal, e depois de três décadas, o BASIC é de longe a linguagem de programação mais popular. *(KURTZ, 1990's)

**(KURTZ. T E. Acervo Universidade de Dortmund. Disponível em: http://cis-alumni.org/BASIC.html.)*

Ada *(Countess of Lovelace)*

O programa de desenvolvimento do projeto Ada, no Departamento de Defesa Americano (DoD) começou em 1975, com o objetivo de estabelecer uma única linguagem de programação de computador de alta ordem apropriada para o DoD em tempo real e para sistemas informáticos embarcados. Um grupo de trabalho de linguagem de Alto Nível (HOLWG) ficou responsável em formular os requisitos do DoD para o projeto, avaliar as linguagens existentes contra aqueles requisitos e implementar o conjunto mínimo de linguagens necessárias para o uso do DoD. Outras partes do esforço incluiu iniciativas administrativas em direção ao objetivo final: especificamente, a Diretiva DoD 5000.29, que prevê que os novos sistemas de defesa devem ser programados em um DoD "aprovado" e centralmente controlado, e a Instrução DoD 5000.31, que deu a lista de definição provisória das linguagens aprovadas.

Os requisitos de linguagem do HOLWG foram amplamente distribuídos para a avaliação das comunidades militares e civis em todo o mundo. Cada versão sucessiva dos requisitos, de STRAWMAN através de STEELMAN, produziu uma definição mais refinada da linguagem proposta. Durante o processo de desenvolvimento de requisitos, determinou-se que o conjunto de atributos gerados era necessário e suficiente para todas as principais aplicações do Departamento de Defesa (e as grandes aplicações). Avaliações formais foram realizadas em dezenas de linguagens existentes. Concluiu-se que nenhuma linguagem existente poderia ser adotada como uma única linguagem comum de alta ordem para o DoD, mas uma única línguagem, atendendo essencialmente a todos os requisitos, era viável e desejável. Quatro contratantes foram financiados para produzir protótipos competitivos. Uma avaliação de primeira fase reduziu os projetos para dois, foram levados para conclusão. Por sua vez, um design de linguagem única foi posteriormente escolhido. Etapas subsequentes incluiu o teste e avaliação da linguagem, controle da linguagem e validação de compiladores. A produção de compiladores e um desenvolvimento de programas e ambiente de ferramentas foram realizados separadamente pelos Componentes de Serviço Individuais.

Os requisitos gerais e expectativas para o ambiente e o controle da línguagem foram abordados em outra série de documentos iterativos. UMA capacidade de validação da linguagem (o conjunto de testes) e recursos associados, foram conformidade com a definição de linguagem de compiladores usando o nome "Ada". O nome Ada foi inicialmente protegida por uma marca comercial do DoD.

(BERGIN. T J e GIBSON. R G, 1996 - adaptado)

C++ *(C with classes)*

O C ++ foi projetado para fornecer as facilidades do Simula para a organização do programa, juntamente com a eficiência do C e flexibilidade para programação de sistemas. O objetivo era entregar isso à projetos reais dentro de 6 meses após a ideia. Foi bem sucedido. Na época, não percebi nem a modéstia nem o absurdo desse objetivo. O objetivo era modesto na medida em que não envolvia inovação, e absurdo, em ambos os tempos e escalas, com suas demandas draconianas para eficiência e flexibilidade. Enquanto uma quantidade modesta de inovação surgiram ao longo dos anos, eficiência e flexibilidade foram mantidas sem compromisso. (STROUSTRUP, 1991)

A pré-história do C ++ ou os anos anteriores à ideia de adicionar recursos semelhantes do Simula para o C, me ocorreu e é importante porque durante esse tempo, os critérios e ideais que posteriormente formaram o C ++ emergiram. Eu estava trabalhando no meu Ph.D. no Laboratório de Computação da Universidade de Cambridge, na Inglaterra. Meu objetivo era estudar alternativas para a organização de software de sistema para um sistema distribuído. A estrutura conceitual foi fornecida pelo computador baseado em recursos Cambridge CAP e seu sistema operacional experimental em constante evolução. Os detalhes deste trabalho e seu resultado são de pouca relevância para o C ++. O que é relevante, no entanto, foi o foco na composição de softwares a partir de módulos bem delimitados e que a principal ferramenta experimental foi um simulador relativamente grande e detalhado que escrevi para simular software rodando em um sistema distribuído. (STROUSTRUP, 1979)

O trabalho, no que eventualmente se tornou C ++, começou com uma tentativa de analisar o kernel do UNIX para determinar até que ponto ele pode ser distribuído por uma rede de computadores e conectados por um rede de área. Este trabalho começou em abril de 1979 no Centro de Pesquisas em Ciência da Computação da Bell Labs em Murray Hill, Nova Jersey, onde tenho trabalhado desde então. Dois subproblemas em breve surgiram: como analisar o tráfego de rede que resultaria da distribuição do kernel e como modularizar o kernel. Ambos exigiam uma maneira de expressar a estrutura do módulo de um sistema complexo e o padrão de comunicação dos módulos. Este era exatamente o tipo de problema que eu havia me tornado determinado a nunca atacar novamente sem ferramentas adequadas. Consequentemente, comecei a desenvolver uma ferramenta de acordo com os critérios que eu havia formado em Cambridge. (BERGIN e GIBSON, 1996)

Linguagens de Scripting

De maneira direta, scripts são “roteiros” seguidos por sistemas computacionais e trazem informações que são processadas e transformadas em ações efetuadas por um programa principal.

Digamos que você precisa de um programa que realize a soma de notas escolares e como elas serão apresentadas. O script é a sequência de passos que o computador vai interpretar para somar e apresentar as notas na tela. Isso também pode ser feito para modificar uma fonte que será exibida em algum site ou programa, repetições etc. Quando você clica em algum botão do seu navegador, a barra com opções é exibida. Parece simples, mas, para que isso aconteça, um script foi ativado no momento em que você clicou com o botão do mouse e desencadeou a exibição do item.

Existem ainda os scripts shell, os quais possuem uma sequência de comando do DOS que são executados quando solicitados. Este tipo de script é normalmente é usado por usuários mais avançados. Existem diversas formas de se criar um script, seja de formatação como de programação. Essas linguagens têm pontos em comum, mas diferem em outros importantes. Algumas das mais comuns para criação de scripts são Java, C++, Lua, Python, entre outras. (PEREIRA - TecMundo, 2012)

Com a maturidade da WEB, o tamanho da comunidade foi a principal força motriz por trás das linguagens de script. Os dias de grandes surpresas e vencedores surpreendentes (como por exemplo a vitória do PHP sobre o Perl no script de websites) acabaram. O sucesso do Python pode explicar parcialmente que o Google patrocinou o idioma, enquanto O'Reilly abandonou o Perl, deixando o desenvolvimento no limbo, devido à complexidade da base de código, que excedeu o tamanho de um projeto de software voluntário. Apesar de serem esforços de código aberto, o desenvolvimento de linguagens de script tornou-se uma área cruel e implacável, governada pela dinâmica impiedosa do mercado. Claro, existem outras linguagens de script valiosas, como REXX, LUA, Erlang, etc, e elas também merecem estudo e podem ser usadas com sucesso em determinados projetos. Mas eu acho que os "big seven" (bash / ksh93 / PowerShell, Javascript, PHP, Perl, Python, Ruby e TCL) estão aqui para ficar. Por favor, note que cada um deles tem um conjunto particular de pontos fortes, o conjunto que o torna excepcionalmente valioso. (BEZRUKOV, 2017)

Javascript

A principal atração do JavaScript é uma sintaxe muito limpa e a capacidade de ser usada tanto como linguagem macro quanto shell (na verdade, o JavaScript é subutilizado como shell no ambiente Windows, apesar de estar disponível por muitos anos via WSH). Também possui modelo de objeto superior. JavaScript é a única linguagem mainstream que usa uma implementação Orientada a Objetos baseada em protótipos (pioneira na linguagem Sun Self). Com linguagens Orientadas a Objetos típicas baseadas em classes, como C ++ e Java, os objetos vêm em dois tipos gerais. As classes são modelos que definem um conjunto de variáveis métodos para o objeto, e as instâncias são classes povoadas - objetos "utilizáveis" com memória alocada e valores de variáveis preenchidas. Eles não podem ser estendidos dinamicamente em tempo de execução.

Na Orinetação a Objetos baseada em protótipo, a estrutura do objeto é dinâmica em tempo de execução e novos objetos são principalmente construídos via clonagem copiando as variáveis e os métodos de um objeto existente (seu protótipo). O novo objeto

pode ser modificado dinamicamente (estendido) sem afetar seu pai. O inverso não é geralmente verdade. Na maioria das implementações Orinetadas a Objetos baseadas em protótipos, o objeto filho mantém um link explícito (via delegação) para o seu protótipo, e as mudanças no protótipo causam mudanças correspondentes em todos os seus clones.

Mas em algumas linguagens Orientados baseados em protótipos não existem essa ligação. Essa é a base do protótipo: definir classes é desnecessário, cada objeto é apenas uma instância dinamicamente estendida do pai. Procedimentos são objetos de dados de primeira classe que podem ser atribuídos a slots do objeto dinamicamente. É como "programar pelo exemplo" em vez de mapear prematuramente o problema em questão no conjunto rígido de classes e depois se arrepender das decisões ingênuas tomadas. (BEZRUKOV, 2017)

PHP

Foi pioneira no nível mais alto de integração com o servidor da Web e o banco de dados. Seu slogan pode ser "A vida é muito curta e é bobagem para gastá-la na resolução de problemas de configuração". Java provou ser mais a parte do problema, do que parte da solução. O PHP foi a primeira linguagem de script que resolveu com sucesso essa necessidade de um conjunto de ferramentas integrado de nível superior para desenvolvedores de sites no Unix. Ele foi projetado explicitamente como parte integrante da troika Webserver / database / scripting_language, geralmente chamada de LAMP. Este provou ser um conjunto de ferramentas muito poderoso, adequado para resolver uma ampla gama de tarefas. É por isso que o PHP depós com sucesso já entrincheirado Perl nesta área particular.

Essa integração perfeita com o banco de dados MySQL e o servidor Apache WEB realmente torna o PHP um importante pioneiro no mundo da linguagem de script. Embora inicialmente fosse uma linguagem de script embutida em HTML do lado do servidor, ela está evoluindo além de suas raízes, HTML para serviços mais avançados, como chamadas de procedimento remoto. (BEZRUKOV, 2017 – adaptado)

APL (*A Programming Lenguage*)

A matemática aplicada se preocupa com o design e a análise de algoritmos ou programas. O tratamento sistemático de algoritmos complexos requer uma linguagem de programação adequada para sua descrição, e tal linguagem de programação deve ser concisa, precisa, consistente em uma ampla área de aplicação, mnemônica e econômica de símbolos; deve exibir claramente as restrições na sequência em que as operações são realizadas; e deve permitir que a descrição de um processo seja independente da representação particular escolhida para os dados. (IVERSON, 1962)

A linguagem APL (A Programming Language) é uma das mais antigas linguagens de programação, tendo sido inventada por Kenneth Iverson em 1957. A primeira implementação da linguagem ficou pronta em 1964.

À semelhança da linguagem Lisp, APL foi idealizada como uma linguagem matemática sem qualquer preocupação com a arquitetura dos computadores onde viria a ser executada. Esta característica que, a princípio, jogava contra a linguagem, permitiu-lhe sobreviver facilmente à evolução do hardware que ocorreu nas últimas quatro décadas existindo hoje várias linguagens (APL2, A+, APL2000, Dyalog APL, Sharp APL, MicroAPL, J, K, Nial, Glee, Nesl e outras) que se auto-proclamam descendentes diretos da

linguagem APL de Iverson. APL caracteriza-se por ser uma linguagem para processamento de arrays. Um array é simplesmente um aglomerado de valores de um mesmo tipo, dispostos numa estrutura retangular. Em termos matemáticos, um array é denominado tensor e possui como casos particulares, os escalares, os vetores e as matrizes. Através da utilização de arrays, é possível operar sobre múltiplos valores em simultâneo, dispensando as estruturas de controle tradicionais (seleção, iteração e recursão) que são necessárias nas linguagens que apenas conseguem lidar com um valor de cada vez. O fato de a linguagem operar sobre vários valores em simultâneo permite também tirar partido da inerente paralelização dessas operações, podendo ser extraordinariamente eficiente em modernas arquiteturas vectorizadas.
(Instituto Superior Técnico - Lisboa. 2005)

Plankalkül *(Plan Calculus)*

Konrad Zuse foi a primeira pessoa na história a construir um computador digital funcional, fato que ainda não é geralmente reconhecido. Ainda menos conhecido é que nos anos 1943-1945, Zuse desenvolveu um modelo de programação de alto nível e, baseado nele, uma linguagem de programação algorítmica chamada Plankalkül (Plan Calculus).

O Plankalkül apresenta tipos de estrutura de dados binários, suportando assim um estilo de programação sem loop para problemas lógicos ou relacionais. Como linguagem para aplicações numéricas, o Plankalkül já possuía as características essenciais de uma “língua von Neumann”, embora ao nível de uma linguagem de operador.

Consequentemente, o Plankalkül é, em alguns aspectos, equivalente e em outros mais poderoso que o modelo de programação de von Neumann que passou a dominar a programação por um longo tempo. Para encontrar conceitos de linguagem semelhantes aos do Plankalkül, é preciso olhar para as "línguas não von Neumann". como o APL ou a álgebra relacional. O Plankalkül não era apenas a primeira linguagem de programação de alto nível, mas em alguns aspectos conceitualmente à frente das linguagens de alto nível que evoluíram uma década depois. *(GILOI. 1997 – adaptado)*

De 1941 a 1946 (ao mesmo tempo em que ele desenvolveu seu computador Z4), Konrad Zuse desenvolveu ideias de como suas máquinas poderiam ser programadas de uma maneira muito poderosa. (isto é, mais poderosa do que cálculos aritméticos somente).

Além de declarações puras para cálculos numéricos, Zuse também usou regras de lógica matemática. Por um lado, ele usou a poderosa lógica de predicados, a álgebra booleana, como construções da linguagem. Por outro lado, ele desenvolveu um mecanismo para definir estruturas de dados poderosas, começando com o bit simples (dígito binário) e trabalhando até estruturas hierárquicas complicadas.

Para demonstrar que a linguagem Plankalkül poderia ser usada para resolver problemas científicos e de engenharia, Konrad Zuse escreveu dezenas de programas para exemplos. Em suas anotações pode-se encontrar a ordenação de listas, estratégias de busca, relações entre pares de listas... Ele usou mais de 60 páginas para descrever programas para jogar xadrez e previu que, em 50 anos, um computador pode vencer o

humano campeão mundial de xadrez. O que se provou ser uma previsão incrivelmente verdadeira.

A linguagem Plankalkül foi descrita inicialmente na dissertação de Ph.D. de Zuse em 1943 e mais tarde desenvolvido em 1945 (também inédito, ainda era tempo de guerra) o trabalho *"Plankalkül. Theorie del angewandten Logistik"* (“Plankalkül. Teoria da logística aplicada”, sigla em alemão) e finalmente chegou ao público em um artigo de 1948 (mas ainda não atraiu muito feedback e por um longo tempo a programação de um computador só seria pensada como programação com código de máquina).

Plankalkül foi finalmente publicado de forma mais abrangente em um artigo de 1972, enquanto o primeiro compilador foi implementado em 1998. (DALAKOV)

*Referencias para o final

(GILOI. WOLFGANG K. (1997). **Konrad Zuse's Plankalkül: The First High-Level, "non von Neumann" Programming Language.** Disponível em: **https://dl.acm.org/citation.cfm?id=612687&preflayout=tabs**. Acesso em 8 de fev. 2019)

(DALAKOV. GEORGI (Last modified: 4 February, 2019). **Plankalkü—the world's first complete high-level language.** Disponível em: https://history-computer.com/ModernComputer/Software/Plankalkul.html. Acesso em 8 de fev. 2019)

(UNIVERSIDADE TÉCNICA DE LISBOA (2005). Disciplina: **Fundamentos da Programação - enunciado de projeto 2005-2006**. p.1)

(IVERSON. KENNETH E. (1962). **A Programming Lenguage**.)

*(KEMENY, John G.; KURTZ, Thomas E. (1985). **Back To BASIC: The History, Corruption, and Future of the Language**. p.1-53.)*

(PEREIRA. ANDRE L. (2012) **O que é script? -** TecMundo. Disponível em: **https://www.tecmundo.com.br/programacao/1185-o-que-e-script-.htm**. Acesso em 8 de fev. 2019)

(BEZROUKOV, NIKOLAI. **Scripting Languages as a Step in Evolution of Very high Level Languages** - All rights reserved. 1998-2017. Disponível em: http://www.softpanorama.org/People/Scripting_giants/scripting_languages_as_vhll.shtml. Acesso em 8 de fev. 2019)