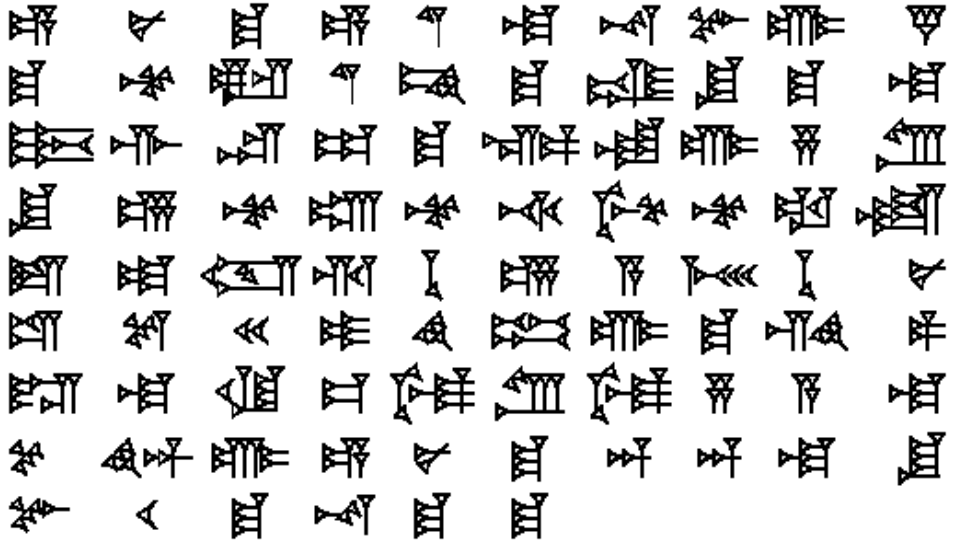
A História das Linguagens de Programação

Prof. Maurício Dias

Slides adaptados de Adam Brooks Webber e da Profa. Rosana Braga

Babilônia

- Escrita cuneiforme: 1790 BC
- Tábuas de argila ainda existem:
  - Poemas e estórias
  - Contratos e acordos
  - Astronomia
  - Matemática



Números da Babilônia

- Base 60. Porque?
- Números de ponto flutuante!

$$1 \times 60^{1} + 10 \times 60^{0} = 70$$

$$1{,}10 = 1 \times 60^{0} + 10 \times 60^{-1} = 1\tfrac{1}{6}$$

$$1 \times 60^{i+1} + 10 \times 60^{i}$$

Programas da Babilônia

- Algoritmos eram escritos usando linguagem natural.
- Algoritmos descreviam procedimentos da vida cotidiana:
  - *Para calcular o volume da cisterna, se seu raio é 2.0 e sua altura é 7.0, então o volume é 3.1 vezes 2.0 vezes 2.0 vezes 7.0*.
- Não usavam variáveis. Números serviam para exemplos.

Baghdad, 780-850

- Enquanto na europa senhores feudais estavam matando uns aos outros, uma cultura rica e vibrante florescia em Baghdad.
- Havia um cortesão e matemático chamado Al-Khorezmi, que escreveu uns livros…

Números indo-arábicos

- O livro dos algoritmos: o original perdeu-se.
- Tradução em latim: *Algorthmi de numero Indorum*.
- Algoritmos para fazer contas com números hindus.
  - Base 10
  - Sistema posicional
- Influenciou fortemente a matemática na Europa medieval.

Outros algoritmos antigos

- Euclides, por volta do ano 300 antes de Cristo, descreveu um algoritmo para calcular MDC.
- Alexander de Villa Dei, 1220 depois de Cristo: Canto de Algorsmo: algoritmos em verso!
- Claro, não havia um método formal para descrever algoritmos. Qualquer coisa valia: linguagem natural, poesia e até música.

Augusta Ada

- Filha de Lord Byron, um grande poeta Inglês.
- 1800 e pouco: mulheres não recebiam educação formal. Matemática era tabu.
- Ada tomou aulas particulares de matemática.
- Casou-se aos 19, tornando-se Lady Lovelace, e teve três filhos.
  - Nenhuma novidade…

Charles Babbage

- Matemático Inglês
- Projetou computadores mecânicos
  - A máquina diferencial (não chegou a ser terminada por Babbage)
  - Máquina analítica (nunca construída)

I wish to God these calculations had been executed by steam!

Charles Babbage, 1821

Ada and Charles

- Ada Lovelace e Charles Babbage ficaram amigos e trabalharam juntos em muitas das idéias de Babbage.
- Ada descreveu um algoritmo para calcular números de Bernoulli usando a máquina diferencial.
  - Não chegou a ser testado, afinal a máquina não existia de fato.
  - Mas é considerado o primeiro programa.

Konrad Zuse

- 1936: construiu um computador mecânico, na sala de estar de seus pais, em Berlim: o Z1
- Era um ábaco mecânico, controlado por pinos de metal e correias.
- Programável via fitas perfuradas.
- Números de ponto flutuante, em binário, com expoente explícito.

Plankalkul

- Em 1945-46, Zuse completou o projeto de uma linguagem de programação: Plankalkul.
- Várias idéias revolucionárias:
  - Atribuição, expressões aritiméticas, subscritos.
  - Tipo primitivo: bit. Tipos derivados: inteiro, real, arranjos, etc.
  - Execução condicional, laços, subrotinas.
  - Asserções!
- Zuse criou vários programas exemplo: ordenação, busca em grafos, análise sintática, etc.

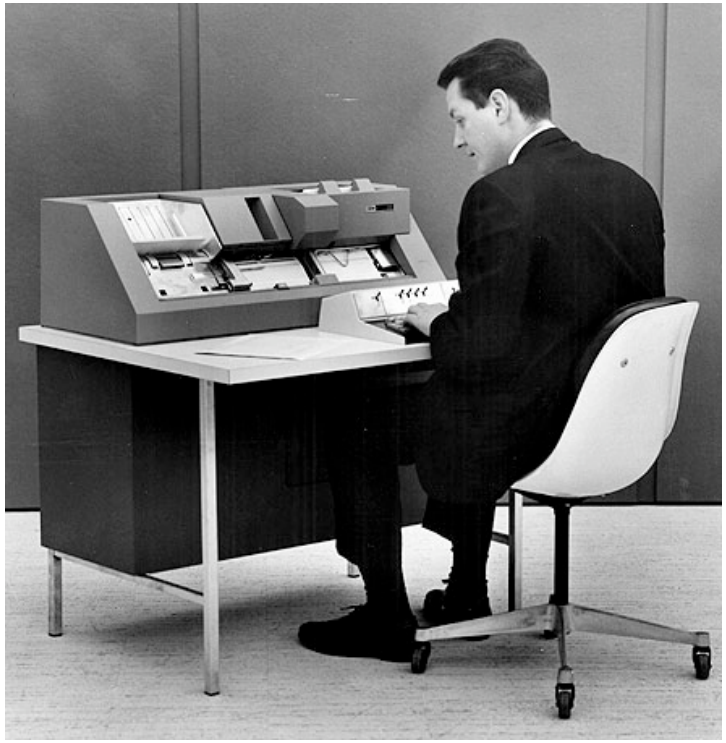
Exemplo

```
P1 max3 (V0[:8.0],V1[:8.0],V2[:8.0]) => R0[:8.0]
max(V0[:8.0],V1[:8.0]) => Z1[:8.0]
max(Z1[:8.0],V2[:8.0]) => R0[:8.0]
END

P2 max (V0[:8.0],V1[:8.0]) => R0[:8.0]
V0[:8.0] => Z1[:8.0]
(Z1[:8.0] < V1[:8.0]) -> V1[:8.0] => Z1[:8.0]
Z1[:8.0] => R0[:8.0]
END
```

- Não chegou a ter influência em outras linguagens. Porque?

Programar dava um trabalho…

- Os programas de AEDs I eram trabalho de semanas, uns 60 anos atrás.
- Programar era trocar fios de lugar.
  - Na melhor das hipóteses, furar cartões.
- É difícil entender quão difícil era esta atividade.



Wish List

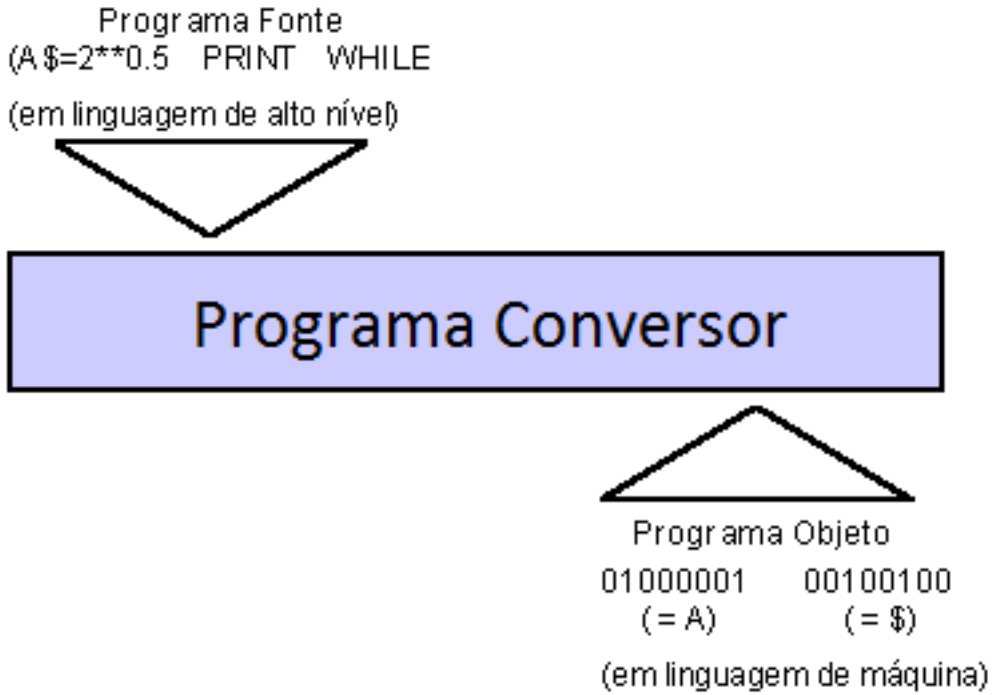
- Números de ponto flutuante: programadores tinham de lembrar qual a posição do ponto.
- Endereço relativo: programadores tinham de saber o endereço das sub-rotinas para computar endereços absolutos.
- Subscritos para arranjos.
- Algo mais simples de lembrar que instruções octais.

As primeiras ferramentas

- Montadores
- Compiladores primitivos:
  - Short code, John Mauchly, 1949
  - A0, A1, A2, Grace Hopper, 1951-53. Compiladores para expressões aritméticas.
  - SpeedCoding, John Backus, 1954
- Era preciso poupar tempo dos programadores.

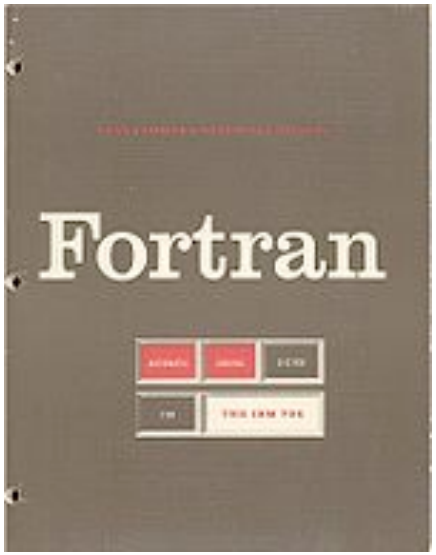
M amknsr_bmp bctc amltcprcp mq amk_lbmq b_bmq ck jgles_eck bc _jrm lđtcjn_p_jgles_eck bc kĆosgl_&aĖbgemq `glĆpgmq',

Cqr_ r_pcd_ bc amltcpqċm č dcgr_ nmp sk npmep_k_ cqncag_j bc amknsr_bmp* gqrm č* sk npmep_k_ oscpcac`c _q glqrpsČĔcq ck jgles_eck bc _jrm lđtcjc bĆ amkmq_đb_ msrpm npmep_k_ amlqrgrsđbm bc glqrpsČĔcq `glĆpg_q, ?m npmep_k_ mpgegl_j* ck jgles_eck bc _jrm lđtcj* bĆ+qc m lmkc bc Npmep_k_ Dmlrc c _m pcqsjr_bm* ck jgles_eck bc kĆosgl_* bc Npmep_k_ M`hcrm,



- INTERPRETADOR
- COMPILADOR
- TRADUTOR

Fortran



- A primeira linguagem de programação a se tornar bastante popular.
- O projeto foi liderado por John Backus, na IBM
  - Era para ter levado seis meses; levou dois anos.
  - Diminuiu os erros de programação
  - Possuia um compilador que gerava código de qualidade.

```
      PROGRAM FIBONA
C
C PROGRAM TO CALCULATE THE SUM OF THE FIRST TEN FIBONACCI NUMBERS -
C   THE RESULT IS HELD IN VARIABLE ANSWER
C
      INTEGER N1, N2, NEW, SUM, ANSWER
C
      N1  = 1
      N2  = 1
      SUM = N1 + N2
C
      DO 10 I=3,10
          NEW = N1 + N2
          N1  = N2
          N2  = NEW
          SUM = SUM + NEW
 10   CONTINUE
C
      ANSWER = SUM
      END
```

Compilação em Separado

- Inicialmente sem compilação modular.
- Porém, programas começaram a crescer.
  - Compilar programas grandes era imprático.
- Com Fortran II veio a possibilidade de compilar módulos, e não programas inteiros.

> I don't know what the language of the year 2000 will look like, but I know it will be *called* FORTRAN.
>
> *C.A.R. Hoare*

Fortran fez escola

- Usada até hoje: previsão de tempo, dinâmica de fluídos, etc.
  - Benchmarks: SPEC CFP 2006.
- O compilador otimizante:
  - Análise léxica
  - Parsing
  - Alocação de registradores

John Backus

- Fortran, Algol 58 e 60, BNF e FP (uma linguagem puramente funcional)

My point is this: while it was perhaps natural and inevitable that languages like FORTRAN and its successors should have developed out of the concept of the von Neumann computer as they did, the fact that such languages have dominated our thinking for twenty years is unfortunate. It is unfortunate because their long-standing familiarity will make it hard for us to understand and adopt new programming styles which one day will offer far greater intellectual and computation power.

John Backus, 1978

BNF (Backus-Naur Form)

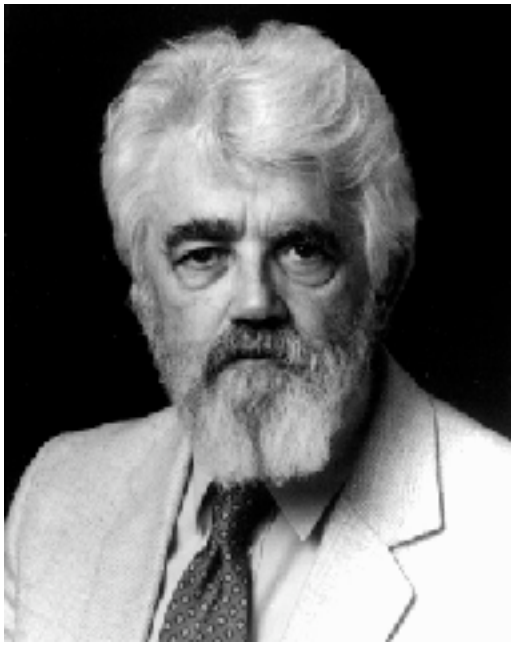
```
(2.0 * PI) / n

<expression>  ::=  <expression> + <term>
              |    <expression> - <term>
              |    <term>
<term>        ::=  <term> * <factor>
              |    <term> / <factor>
              |    <factor>
<factor>      ::=  number
              |     name
              |     ( <expression> )
```

LISP

- Em 1956 aconteceu uma conferência de IA em Dartmouth: McCarthy, Minksy, Newell, Simon.
- Newell, Shaw e Simon introduziram *Logic Theorist*, um programa de racioncínio escrito em IPL (Information Processing Language).
- IPL tinha suporte a listas encadeadas, e chamou a atenção de McCarthy.
- Surgia LISP, a primeira linguagem funcional.

Uma linguagem para IA

- John McCarthy era um professor no MIT, trabalhando com IA.
- Ele tinha uma lista de desejos:
  - Expressões condicionais
  - Recursão
  - Funções de alta ordem (como **`map`**)
  - Coletor de lixo.
- Fortran não servia…



A sintaxe de LISP

- Um programa é uma lista, representando uma AST: (+ a (* b c))
- McCarthy escreveu uma função eval para interpretar a AST.
  - A AST acabou virando a própria linguagem.
- Esta função eval foi escrita a mão, usando linguagem *assembly*.

```
;; This example function draws a line with a Linetype Scale of 0.5
;; between two points selected by the user.

(defun c:DrawLine ( / p1 p2 )

  ;| Example penned by Lee Mac 2011
     www.lee-mac.com |;

  (if
    (and
      (setq p1 (getpoint "\nSpecify First  Point: "   ))
      (setq p2 (getpoint "\nSpecify Second Point: " p1))
    )
    (entmakex
      (list
        (cons 0 "LINE")
        (cons 10 (trans p1 1 0))
        (cons 11 (trans p2 1 0))
        (cons 48 0.5)
      )
    )
  )
  (princ)
)
```

A evolução de LISP

- LISP é possivelmente a linguagem mais popular para IA.
- Até por volta de 1980 havia muitos dialetos:
  - Cada grupo de pesquisa em IA tinha *seu* próprio LISP.
  - Havia inclusive computadores que foram desenvolvidos somente para executar LISP.
- Hoje há uma certa padronização:
  - Common LISP: a linguagem e bibliotecas.
  - Scheme: um dialeto mais simples, ensinado em escolas.

A influência de LISP

- LISP é a segunda linguagem de programação ainda em uso.
- Idéias como expressões condicionais e recursão são amplamente adotadas hoje.
- Muitas linguagens funcionais surgiram.
- A coleta de lixo é também muito popular.

Algol

- Em 1957 as linguagens de programação estavam surgindo aos montes.
  - Indústrias tinham seus padrões.
  - Universidades tinham seus padrões.
  - Havia muitos padrões, e nenhuma padronização.
- Algol surgiu para acabar com isto.
  - Universal, independente de máquina.
- Um comitê internacional foi criado em 1958, para estabelecer o projeto.

Muitos Algols (!)

- No final das contas, três projetos: Algol 58, Algol 60 e Algol 68.
- Os comitês foram ficando cada vez maiores e mais estrelados.

```
«integer procedure gcd(m, n) ;
    value m, n ;
    integer m, n ;
    comment computes greatest common divisor of m and n ;
    begin
        integer c ;
        m := abs(m) ;
        n := abs(n) ;
        if n=0 then goto zero ;
div:    c := entier(m/n) ;
        m := m - c×n ;
rep1:   if m≧n then begin m := m - n ; goto rep1 end ; [1]
rep2:   if m<0 then begin m := m + n ; goto rep2 end ; [1]
        c := m ;
        m := n ;
        n := c ;
zero:   if n=0 then gcd := m else goto div
    end gcd».
```

A longa herança

- Quase toda linguagem que surgiu depois de 1958 usa idéias de Algol:
  - Blocos delimitadores.
  - Estrutura léxica de formato livre.
  - Sintaxe definida via BNF
  - Escopo de bloco para variáveis locais
  - Tipagem estática com anotações de tipo
  - If-then-else's aninhados
  - Chamada por valor
  - Recursão e expressões condicionais
  - Alocação dinâmica de memória.
  - Procedimentos de primeira classe.
  - Operadores definidos pelo usuário.

Polêmicas

- As primeiras linguagens usavam rótulos e go-to's para criar desvios de fluxo.
- Algol e similares vieram com uma nova proposta, baseada em estruturas de controle.
- Houve muita polêmica.
- Em 1968, Edsgar Dijkstra escreveu um artigo muito famoso: "*Go to statement considered harmful*".

A Programação Estruturada

- A programação baseada em estruturas de controle, em vez de rótulos, é chamada *programação estruturada*.
- Muitos programadores achavam difícil programar sem go-tos.
- Hoje a polêmica parece ter chegado ao final
  - Algumas linguagens, como Java, nem possuem go-to's
  - E mesmo em linguagens onde go-to's existem, os programadores raramente os usam.
- Esta polêmica toda surgiu com Algol ☺

Ortogonalidade

- O projeto de Algol tentava ao máximo evitar casos especiais:
  - Sintaxe independende da forma dos programas
  - Eliminação de limites arbitrários: nomes podem ter qualquer tamanha, arranjos podem ter qualquer número de dimensões, etc.
  - Ortogonalidade: conceitos podem ser combinados de qualquer forma. Ex.: declaração de parâmetros = declaração de variáveis, arranjos de qualquer tipo primitivo, registros com quaisquer campos, etc.

Exemplo de Ortogonalidade

	Integers	Arrays	Procedures
Passar como parâmetro	x	x	x
Armazenar em variável	x	x	x
Armazenar em arranjo	x	x	x
Retornar de uma função	x	x	x

- Cada combinação proibida é um caso especial, que precisa ser lembrado pelo programador.
- Em Algol 68 todas as combinações acima são possíveis.
- Poucas linguagens modernas levam ortogonalidade tão a sério quanto Algol.

Más notícias

- Algol nunca foi tão usada quanto se esperaria.
  - Algol 58 deu origem a *Jovial*.
  - Algol 60 foi usada como padrão de publicação de algoritmos.
- Razões do insucesso:
  - O projeto negligenciou entrada/saída.
  - A linguagem era considerada complicada.
  - Alguns erros: passagem por nome (!)
  - Não havia suporte coorporativo ou governamental.

Os grandes problemas da programação imperativa, estruturada:

-- Grande Acoplamento!

-- Baixa Coesão!

Temos os dados, e o programa é constituído por milhares de funções que…

-- Ou manipulam diretamente esses dados

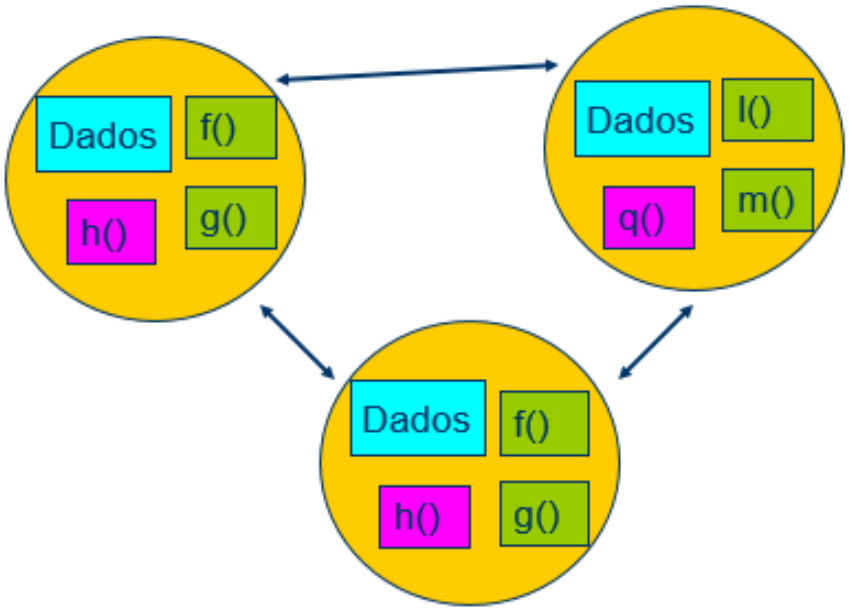
-- Ou trocam muitos valores por parâmetro

Em OOP (Object-Oriented Programming), as funções estão encapsuladas juntamente com os dados a que podem (e devem acessar)

A principal ideia dos objetos é que:

- Apenas as funções relacionadas com os dados os acessam
- Reduzir o acoplamento e aumentar a coesão, isto é, permitir a construção de software em projetos de larga escala, de forma consistente e fácil de gerenciar
- Além disso, é muito mais natural pensar em termos de objetos e suas relações do que em termos de dados e algoritmos

Programação estruturada:
PROGRAMA = DADOS + ALGORITMOS
Programação orientada a Objetos
PROGRAMA = OBJETOS + RELAÇÕES



Havia Simula…

- Havia uns navios entrando no porto da Noruega.
- Kristen Nyggard and Ole-Johan Dahl tinham de simular estes navios.
- Simula I: uma variante de Algol, com extensões para facilitar simulações: aviões no aeroporto, clientes no banco, etc.
- Simula 67: uma linguagem de propósito geral com classes, objetos e herança.

SmallTalk

- Alan Kay, Xerox, 1972
- Smalltalk foi inspirada por Simula, Sketchpad, Logo, biologia celular, etc.
- Smalltalk é mais orientada por objetos que a maior parte de seus descendentes.
- Tudo são objetos: variáveis, constantes, classes, registros de ativação, etc.
- Toda computação é feita por objetos que recebem e emitem mensagems: 1 + 2 = 1.sum(2)

```
checkOut
    self age value < 18
        ifTrue:
            [self orderApproval
                ifTrue:
                    [self orderManager processOrder: self shoppingBag list.
                     Dialog warn: 'Your order was checked out'.]
                ifFalse:
                    [self disableCheckOutButton.
                     Dialog warn: 'Parental approval required'.
                     self enableApprovalForm]]
        ifFalse:
            [self orderManager processOrder: self shoppingBag list.
             Dialog warn: 'Your orded was checked out']
```

Filosofia de projeto

- Crie a linguagem em torno de idéias simples:
  - Listas, recursão, eval: LISP
  - Objetos, troca de mensagens: Smalltalk
  - Inferência e unificação: Prolog
- Benefícios:
  - A implementação inicial é simples
  - É fácil modificar a linguagem depois
  - A curva de aprendizado é pequena.

A Influência de Smalltalk

- Juntamente com Simula, Smalltalk influenciou uma geração de linguagens orientadas por objetos.
- Smalltalk ainda é usada.
- Muitas linguagens OO acabaram enfatizando a eficiência de compilação.
  - Tipagem estática (smalltalk é dinâmica)
  - Tipos primitivos que não são objetos.

Prolog

- Em 1965 Alan Robinson estava trabalhando em um provador de teoremas.
  - Provas eram encontradas via unificação.
- Em 1971 (Edinburgh) Robert Kowalki desenvolveu técnicas de resolução mais eficientes.
- Alain Colmerauer e Philippe Roussel inventaram Prolog em 1972.
  - Prolog era parte de um projeto em IA (Marseilles, França), baseado em deduções lógias.


file5.pro
2:15
Insert
Indent
/*
father("Bill","John").
father("Pam","Bill").
*/
father(person("Bill","male"),person("John","male")).
father(person("Pam","female"),person("Bill","male")).
grandFather(Person,GrandFather):-
father(Father,GrandFather),
father(Person,Father).

A evolução de Prolog

- Uma nova versão em 1973
  - O programador tinha a opção de parar o *backtracking*.
  - O teste de ocorrência acabou sendo eliminado.
- David warren inventou um complador eficiente e m 1977, baseado nas *Máquinas Abstratas de warren* (intermediária sequencial)
- Estas técncias de compilação acabaram sendo usadas em outras linguagens: Smalltalk e ML, por exemplo.

ML

- Provadores de teorema estavam na moda.
- Robin Milner, Edingurgh, 1974
- LCF: uma ferramenta para desenvolver a construção de provas formais de teoremas.
- ML foi projetada como a linguagem usada em LCF para escrever axiomas e teoremas.
- Tipagem forte, polimorfismo paramétrico e inferência de tipos já estavam no projeto.

Semântica Formal

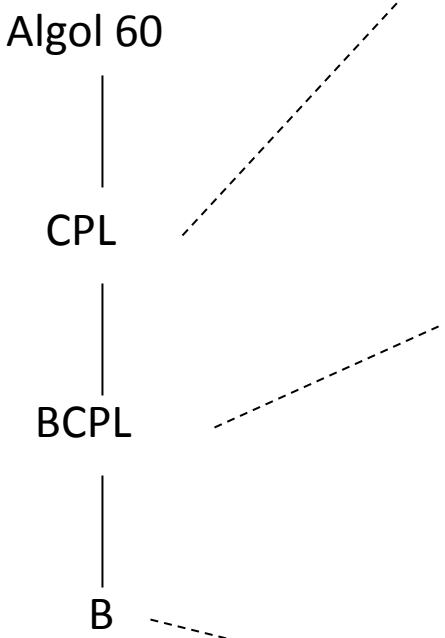
- A definição de ML inclui sua semântica formal
  - Semântica natural.
- Em geral, a linguagem aparece antes de sua semântica forma.
  - Em ML foi o contrário.
- Foi assim porque, a fim de confiar nas provas de LCF, era preciso ter certeza na corretude de ML.

A evolução de ML

- Em 1980 Luca Cardelli implementou um compilador eficiente para ML.
- Em 1983 foi publicado o padrão de ML.
- Novas construções: casamento de padrões, módulos, registros, tratamento de exceções.
- Dialetos:
  - Standard ML (a versão que usamos)
  - Lazy ML: ML com avaliação preguiçosa
  - Caml: Um dialeto criado antes da adição de módulos
  - Ocaml: Caml com orientação por objetos.

```
fun factorial n = let
  fun fac (0, acc) = acc
    | fac (n, acc) = fac (n - 1, n * acc)
  in
    if (n < 0) then raise Fail "negative argument"
    else fac (n, 1)
  end
```

Os descendentes de Java



Uma linguagem ainda maior que Algol, com construtos para processamento de dados de negócios. Christopher Strachey et al. 1962-66

"Basic CPL." CPL simplificada. Sem tipos: dados são palavras de máquina. Introduziu a notação comum de arranjos: A[i] e chaves para blocos.
Martin Richards (a student of Strachey), 1967

Uma linguagem ainda mais simples que BCPL. Foi usada nos primeiros sistemas UNIX. Re-introduziu as atribuições compostas: (`a+=b`), vistas em Algol 68.
Ken Thompson, 1969

CPL

```
Max(Items, ValueFunction) = value of
§ (Best, BestVal) = (NIL, -∞)
while Items do §
(Item, Val) = (Head(Items), ValueFunction(Head(Items)))
if Val > BestVal then (Best, BestVal) := (Item, Val)
Items := Rest(Items) §□
result is Best §□
```

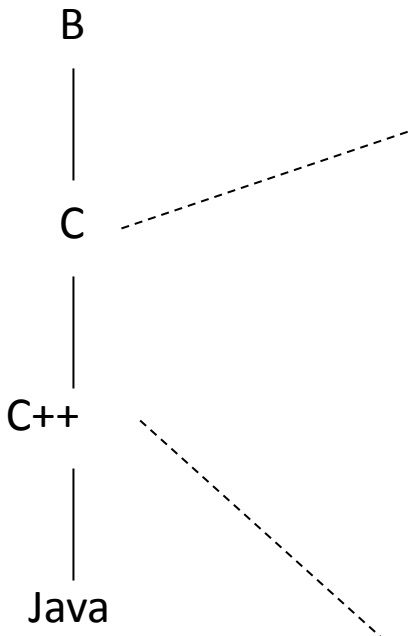
BCPL

```
GET "LIBHDR"

LET START () BE
$(
   WRITES ("Olá Mundo!*N")
$)
```

B

```
/* The following function will print a non-negative number, n, to
   the base b, where 2<=b<=10.  This routine uses the fact that
   in the ASCII character set, the digits 0 to 9 have sequential
   code values.  */

printn(n, b) {
        extrn putchar;
        auto a;

        if (a = n / b)          /* assignment, not test for equality */
                printn(a, b); /* recursive */
        putchar(n % b + '0');
}
```

Mais descendentes



Extensão de B (originalmente chamada "NB"). O objetivo era aproveitar melhor o hardware (PDP-11). Sistema de tipos, pre-processador, bibliotecas de entrada/saída, etc. Foi usada para re-implementar o kernel do UNIX e várias aplicações deste sistema operacional.

Dennis Ritchie et. al., 1971-1973

C++ era originalmente um preprocessador de C que incluia na linguagem orientação por objetos: "C++ = C com Classes". Trouxe dynamic dispatch, sobrecarga de operadores e funções, polimorfismo paramétrico, tratamento de exceções.

Bjarne Stroustrup, 1984

C

A primeira versão de C foi criada por Dennis Ritchie em 1972 nos laboratórios Bell para ser incluído como um dos softwares a serem distribuídos juntamente com o sistema operacional Unix do computador PDP-11, na equipe certificada por Ken Thompson.

Tanto BCPL quanto B mostravam-se muito limitadas, prestando-se apenas para certas classes de problemas. Isto se fez sentir especialmente na primeira versão do PDP11, lançado no mercado em 1971. Um dos fatores que levou à isto foi a intenção do grupo responsável pelo UNIX de reescrevê-lo todo em uma linguagem de alto nível, e para isto B era considerado lenta.

Estes problemas levaram a que o projetista Dennis Ritchie, do Bell Labs, fosse encarregado de projetar uma nova linguagem, sucessora do B, que viria então, a ser chamada de C. A linguagem C buscou manter o "contato com o computador real" e ainda sim dar ao programador novas condições para o desenvolvimento de programas em áreas diversas, como comercial, científica e de engenharia.

C++

Bjarne Stroustrup queria ter classes e objetos na linguagem C e Criou um pré-processador que compilava a sua linguagem “C with Classes” para C
1984, Bell Labs, C++

Alguns dos problemas do C++ é que é muito grande, complicada de utilizar e muito fácil de cometer erros/gerar código de baixa qualidade pois depende da qualidade do programador muito mais do que depende de ferramentas da linguagem

Java

- James Gosling, Sun Microsystems
- 1991: Oak – uma linguagem para aplicações de rede.
  - Parecida com *C++*, porém menor e mais simples.
  - Mais segura – fortemente tipada.
  - Mais portável – máquina virtual.
- Em 1995 foi renomeada Java, e voltou-se para a web.
  - Incorporada em navegadores.

Filho de muitos pais

- Java não vem somente de CPL.
- Java também tem:
  - Coleta de lixo, como LISP.
  - Concorrência, como Erlang.
  - Pacotes, como Modula
- Mas java não tem nada novo. Seu objetivo é ser uma linguagem de produção, não uma linguagem de pesquisa.

A calçada da fama

- Muitos pioneiros das linguagens de programação ganharam Prêmios Turing:
  - Alan Perlis, John McCarthy, Edsger Dijkstra, Donald Knuth, Dana Scott, John Backus, Robert Floy, Denneth Iverson, C.A.R. Hoare, Dennis Ritchie, Nickaus Wirth, John Cocke, Robin Milner, Kristen Nyggard, Ole-Johan Dahl
- Foi esta gente que popularizou coisas que hoje são óbvias:
  - Variáveis locais com escopo de bloco.
  - Programação estruturada.

E o futuro?

- Será que ainda há espaço para mais evolução entre as linguagens de programação?
- Talvez todas as descobertas importantes já foram feitas, e a evolução agora será muito mais lenta.
- Ou talvez ainda teremos o prazer de assistir o surgimento de novas idéias, hoje desconhecidas, mas que se tornaram óbvias para todo o mundo.