

Paradigmas de Programação

Prof. Maicon R. Zатели

Aula 1 - Panorama Histórico

Universidade Federal de Santa Catarina
Florianópolis - Brasil

Panorama Histórico - O Computador

Ideia do computador parte dos conceitos de:

- Máquina programável (Ada Lovelace)
- Lógica combinatória (Schonfinckel - Haskell B. Curry)
- Computabilidade (Church - Turing)
- Arquitetura de máquina de estado com memória e endereço (Modelo von Neumann)

Panorama Histórico - Máquina programável

Cartões da máquina de bordados de Jacquard

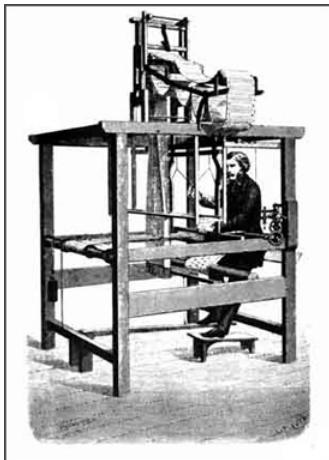
(<https://www.youtube.com/watch?v=0lJns3fPItE>).

Babbage e Ada: inventaram a máquina analítica a partir do tear de Jacquard.

- Substituíram os cartões de operação (de desenho do jacquard) por cartões de padrões algébricos.

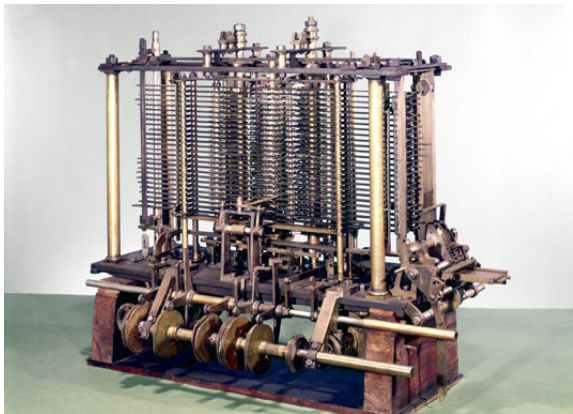
Ada criou os cartões para programar a máquina, tornando-se a primeira programadora.

Panorama Histórico - Tear de Jacquard



Fonte: http://docencia.fca.unam.mx/arojas/html_interiores/jac_char.htm

Panorama Histórico - Máquina analítica



Fonte: <http://estoriasdahistoria12.blogspot.com/2012/12/197-aniversario-de-ada-lovelace.html>

Panorama Histórico - Lógica combinatória

Notação introduzida por Moses Schonfinckel, revista e ampliada por Haskell Curry, para eliminar a necessidade de variáveis na lógica matemática.

São transformações por meio de funções de alta ordem descritas previamente, chamados de combinadores SKI.

Panorama Histórico - Lógica combinatória

I retorna o argumento (identidade): $Ix \rightarrow x$

K quando aplicado a qualquer argumento x , origina uma função constante a um argumento Kx , que quando aplicado a qualquer tese, retorna x : $Kxy \rightarrow x$

S é um operador de substituição. Leva três argumentos e, em seguida,

- retorna o primeiro argumento aplicado ao terceiro, o qual é então
- aplicado ao resultado do segundo argumento aplicado ao terceiro,

ou seja, $Sxyz \rightarrow xz(yz)$

Panorama Histórico - Lógica combinatória

① SKSK

Panorama Histórico - Lógica combinatória

- 1 SKSK
 - Usando a regra-S: $KK(SK)$

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

2 SII_{α}

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

2 SII_{α}

- Usando a regra-S: $I_{\alpha}(I_{\alpha})$

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

2 SII_{α}

- Usando a regra-S: $I_{\alpha}(I_{\alpha})$
- Usando a regra-I: $I_{\alpha}\alpha$

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

2 $SII\alpha$

- Usando a regra-S: $I\alpha(I\alpha)$
- Usando a regra-I: $I\alpha\alpha$
- Usando a regra-I: $\alpha\alpha$

Panorama Histórico - Lógica combinatória

1 SKSK

- Usando a regra-S: $KK(SK)$
- Usando a regra-K: K
- Fim. Nenhuma outra regra pode ser aplicada.

2 $SII\alpha$

- Usando a regra-S: $I\alpha(I\alpha)$
- Usando a regra-I: $I\alpha\alpha$
- Usando a regra-I: $\alpha\alpha$
- Fim. Nenhuma outra regra pode ser aplicada.

Panorama Histórico - Computabilidade

Alonso Church: usando lógica combinatória desenvolve o cálculo lambda e usando cálculo- λ prova a computabilidade.

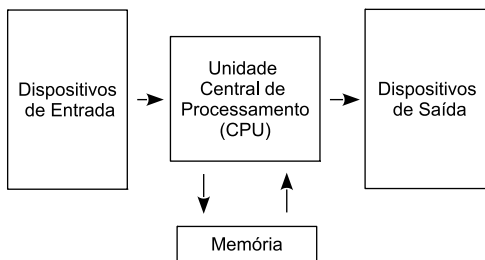
Alan Turing: usa um modelo de máquina de estado para provar a mesma coisa, a Máquina de Turing.

- Qualquer coisa que pode ser computável por uma Máquina de Turing, pode ser chamada de computável. O mesmo vale para cálculo- λ .
- Qualquer computação que pode ser executada por meios mecânicos pode ser executada por uma Máquina de Turing.

Panorama Histórico - Modelo de von Neuman

Proposto por John von Neumann em 1940.

O Processador segue as instruções armazenadas em uma memória de programas, para ler canais de entrada, enviar comandos sobre canais de saída e alterar as informações contidas em uma memória de dados.



Panorama Histórico

IF			10	0F		00	
							EAX
							EBX
							ECX
							EDX
							ESP
							EBP
							ESI
							EDI
							EIP
							EF
							CS
							SS
							DS
							ES
							FS
							GS

Panorama Histórico - Linguagens

Linguagem de máquina

10100000 00110110001110101...

Código: A0h; Instrução: mov AX, moffset

Endereço de memória de 64 bits

O processador:

- 1 Lê a instrução apontada pelo registrador IP (Instruction Pointer).
- 2 Carrega a instrução A0h no reg. CS (Code segment).
- 3 Carrega os 64 bits de moffset no reg. DS (Data Segment) ou 32+32 nos registradores DS:ES.
- 4 Transfere 64/32 bits no endereço de memória apontado pelo reg. para o reg. AX e incrementa o reg. IP.

Panorama Histórico - Linguagens

rótulo: mnemônico, argumento1, argumento2, argumento3 ...

- O rótulo é um identificador seguido por :
- O mnemônico é uma palavra reservada para o código da instrução do processador
- O número de argumentos depende da instrução e do código do processador

Exemplo

```
soma: add edx, 10
```

Panorama Histórico - Linguagens

Difícil montar programas diretamente no set de instruções do processador: Assembly, autocode, IPL (codigos simbolicos).

Assembly foi criado para facilitar a montagem do programa (assembler = montador).

Assembly não é propriamente uma linguagem

Ideia: linguagem de alto nível -> código objeto -> montador

Solução: **Compiladores**

Panorama Histórico - Linguagens - FORTRAN

FORTRAN (FORmula TRANslator) - 1954

Procedural e imperativa

Criada pela IBM (John Backus)

Dedicada a resolução de equações e fórmulas matemáticas

FORTRAN II: loops, funções, sub-rotinas e a primitiva do comando FOR.

Panorama Histórico - Linguagens - FORTRAN 77

```
program fatorial
  implicit none
  integer N
  parameter (N=5)
  integer i
  integer produto

  produto = 1

  do i = 1,N,1
    produto = produto * i
  end do

  write(*,*) 'fatorial de ', N, ' = ', produto

end
```

“implicit none” indica que todas as variáveis devem ter seu tipo pré-definido

Tente executar este código em:

https://www.tutorialspoint.com/compile_fortran_online.php

Panorama Histórico - Linguagens - LISP

LISP (LISt Processor) - 1958

Funcional

Criada por John McCarthy

Desenvolvida para processamento de listas

Puramente recursiva e não iterativa

Não diferencia código e dados

Panorama Histórico - Linguagens - LISP

```
(defun fatorial (n)
  (if (= n 0)
      1
      (* n (fatorial (- n 1)))
  )
)

(defun main()
  (write-string (write-to-string (fatorial 5)))
)

(main)
```

Tente executar este código em:

http://www.compileonline.com/execute_lisp_online.php

Panorama Histórico - Linguagens - ALGOL

ALGOL (ALGOritmic Language) - 1958

Procedural, criada em 58 como IAL (International Algorithmic Language)

Criada por comitê de especialistas em computação

Primeira linguagem autônoma, independente de arquitetura (portável)

Introduziu a declaração em blocos e variáveis locais, arrays dinamicos, := para atribuição, loops IF...THEN...ELSE, FOR, SWITCH, WHILE

ALGOL 68 define cast de tipos e UNION.

Panorama Histórico - Linguagens - ALGOL68

```
BEGIN INT n = 5;  
  BEGIN  
    INT produto := 1;  
    FOR i TO n DO  
      produto *:= i  
    OD;  
    print (("fatorial de ", n, " = ", produto))  
  END  
END
```

Tente executar este código em:

http://www.compileonline.com/execute_algol_online.php

Panorama Histórico - Linguagens - BASIC

BASIC (Beginners All-purposes Symbolic Instruction Code) - 1963

Procedural

Criada por John Kemeny e Thomas Kurtz.

Originalmente, código de linhas numeradas e subrotinas chamadas por linha (GOTO e GOSUB)

Panorama Histórico - Linguagens - BASIC

```
100 LET X = 5
110 GOSUB 300
120 PRINT "fatorial de "; X; " = "; F
130 END
300 LET F = 1
310 FOR I = 1 TO X
320     LET F = F * I
330 NEXT I
340 RETURN
```

Tente executar este código em: <http://www.quitebasic.com/>

Panorama Histórico - Linguagens - C

C (nome dado depois da linguagem B) - 1965

Procedural (da linhagem da ALGOL)

Criada por Brian Kernighan e Denis Ritchie.

Linguagem destinada para programar sistemas Unix a partir do BCPL (1965) e B (1967) desenvolvidas pela AT&T foi padronizada em 1973 (ANSI C).

Conceito de blocos, bibliotecas (headers) de funções, array, pointers e casting de tipos.

É uma das linguagens mais utilizada até hoje.

Panorama Histórico - Linguagens - C

```
#include <stdio.h>

int main() {
    int n = 5;
    int fatorial = 1;
    int i;

    for (i = 1; i <= 5; i++) {
        fatorial *= i;
    }
    printf("fatorial de %d = %d \n", n, fatorial);

    return 0;
}
```

Tente executar este código em:

http://www.compileonline.com/compile_c_online.php

Prolog (logic programming) - 1972

Lógica

Criada por Alain Colmerauer e Philippe Roussel.

Linguagem puramente lógica e baseada num subconjunto do cálculo de predicados de primeira ordem, o que é definido por cláusulas de Horn. Alguns conceitos fundamentais são unificação, recursão, e backtracking.

Panorama Histórico - Linguagens - Prolog

```
:- initialization(main).  
fatorial(1,1).  
fatorial(N,K) :- N1 is N - 1, fatorial(N1, K1), K is N * K1.  
main :- write('fatorial de 5 = '), fatorial(5, X), write(X).
```

Tente executar este código em:

http://www.compileonline.com/execute_prolog_online.php

Panorama Histórico - Linguagens - Haskell

HASKELL (nome em homenagem a Haskell Curry) - 1990

Funcional (diretamente derivada da ML)

Criada pela Universidade de Glasgow.

Linguagem puramente funcional e baseada em cálculo lambda tipado.

Panorama Histórico - Linguagens - Haskell

```
fatorial 0 = 1
fatorial n = n * fatorial (n-1)

main = putStrLn ("fatorial de 5 = " ++ show (fatorial 5))
```

Tente executar este código em:

http://www.compileonline.com/compile_haskell_online.php

Panorama Histórico - Linguagens

Linguagens: 1970 - 2018

Várias linguagens derivadas das anteriores:

Imperativas: Forth - Modula 2 - Perl - PHP - Javascript - C# ...

Orientadas a Objetos: Smaltalk - ADA - C++ - Eiffel - Python - Ruby - Java ...

Declarativas: Prolog - SQL - HTML - UML - XML - Scheme - ML - Miranda - Haskell - OHaskell (OOP) - Clean - CAML - OCAML (OOP) - UML ...

Panorama Histórico - Linguagens - Assembly (NASM)

Objetivo é gerar o mesmo código... (parte 1)

Inicialização

```
section .data
    msg db 'fatorial de 5 = ',0xa
    len equ $ - msg           ;tamanho da mensagem
    num db '                   ',0xa ;variavel para armazenar resultado

section .bss
    lennum resb 1

section .text
    global _start
_start:
    mov edx, len              ;tamanho da mensagem
    mov ecx, msg              ;mensagem para escrever (endereço)
    mov ebx, 1                ;file descriptor (stdout)
    mov eax, 4                ;system call number (sys_write)
    int 0x80                  ;call kernel

    mov eax, 1                ;armazena o produto (fatorial)
    mov ecx, 5                ;fatorial de 5
```

Panorama Histórico - Linguagens - Assembly (NASM)

Objetivo é gerar o mesmo código... (parte 2)

Calcula o fatorial

```
fatorial:
    mul ecx          ;eax *= ecx
    dec ecx          ;ecx--
    cmp ecx, 0       ;ecx == 0
    jne fatorial     ;if (ecx != 0) goto fatorial

    ;conversao de numero para ASCII
    mov ebx, 11       ;tamanho do vetor "num"
    mov [lennum], ebx;salva o tamanho do vetor "num" em lennum
    mov ebx, num+9     ;comeca a armazenar o numero da direita par esquerda
    mov edi, 10        ;divisor 10
```

Panorama Histórico - Linguagens - Assembly (NASM)

Objetivo é gerar o mesmo código... (parte 3)

Converte o resultado para ASCII

```
toascii:
    mov edx, [lennum];
    dec edx           ;lennum--
    mov [lennum], edx;

    mov edx, 0        ;limpa para salvar o resto
    div edi           ;eax /= 10 e edx = eax % 10
    add edx, '0'      ;converte o resto para ascii
    mov [ebx], dl     ;salva o resto no vetor "num"
    dec ebx           ;move ponteiro para esquerda
    cmp eax, 0        ;eax == 0
    jne toascii       ;if (ecx != 0) goto toascii
```


Panorama Histórico - Linguagens - Assembly (NASM)

Objetivo é gerar o mesmo código... (parte 4)

Imprime o resultado

```
inc ebx          ;aponta para a posicao inicial do numero
mov edx, [lennum];tamanho da mensagem
mov ecx, ebx     ;mensagem para escrever (endereço)
mov ebx, 1       ;file descriptor (stdout)
mov eax, 4       ;system call number (sys_write)
int 0x80         ;call kernel

mov eax, 1       ;system call number (sys_exit)
int 0x80         ;call kernel
```

Tente executar o código completo em:

http://www.compileonline.com/compile_assembly_online.php

Linguagens de Programação

Linguagem de programação é uma linguagem artificial usada para facilitar o controle de uma máquina de estados, principalmente para a especificação exata de algoritmos. É destinada a descrever o conjunto das ações consecutivas que um computador deve executar.

Enquanto um **programa** é uma sequência de passos (finita) escritos em uma **linguagem de programação** (entendidas por computadores), um **algoritmo** é uma sequência de passos (finita) escritos em linguagem natural (entendida por nós, humanos).

Linguagens de Programação - Programa

Bellman-Ford

```
void bellmanFord(int s) {
    int u, v;
    for (v = 1; v <= V; v++) {
        d[v] = INF;
        pi[v] = NIL;
    }
    d[s] = 0;
    for (int i = 0; i < V; i++) {
        for (u = 1; u <= V; u++) {
            for (int j = 0; j < pos[u]; j++) {
                v = adj[u][j];
                if (d[v] > d[u] + w[u][v]) {
                    d[v] = d[u] + w[u][v];
                    pi[v] = u;
                }
            }
        }
    }
}
```

Bellman-Ford

```
Initialize()
Bellman-Ford(G,W,s)
  for i = 1 to |V| - 1
    for each edge (u, v)  $\in$  E
      Relax(u, v)
```

Linguagens de Programação

Paradigma de programação é uma maneira de se programar a solução de um problema de forma que possa ser expresso por uma linguagem.

- Imperativos ou declarativos
- Procedurais ou funcionais
- Estruturado ou orientado a objetos

As linguagens de programação podem suportar mais de um paradigma. Ex: Python.

Linguagens de Programação

Linguagens de programação são formadas por:

- Componentes léxicos
- Regras de sintaxe
- Estruturas semânticas

Linguagens de Programação

Componentes léxicos

- Palavras: nomes, comandos, variáveis, etc.
- Símbolos: aspas, parênteses, #, \$, +, -, etc.
- Pontuações: pontos, vírgulas, pontos e vírgulas, etc.
- Brancos: espaços, tabulações, quebras de linha, etc.

Linguagens de Programação

Regras de sintaxe

- Combinações de palavras, brancos, símbolos e pontuação.

Gramática - BNF

```
<expr>    ::= <expr> "+" <term> | <expr> "-" <term> | <term>  
<term>    ::= <term> "*" <factor> | <term> "/" <factor> | <factor>  
<factor>  ::= "(" <expr> ")" | identifier | number
```

Exemplo

(a + 5) / 5 * 2

* BNF é uma notação definida por Peter Naur e melhorada por John Bakus. Por isso o nome Backus-Naur Form (BNF).

* EBNF (Extended-BNF) inclui novos símbolos, como *, ?, e +.

Linguagens de Programação

* EBNF (Extended-BNF) inclui novos símbolos, como *, ?, e +.

- * o símbolo (ou grupo de símbolos entre parênteses) à esquerda do operador * é repetido **zero ou mais** vezes.

```
<declaration> ::= <type> identifier ("," identifier)* ";"
```

- + o símbolo (ou grupo de símbolos entre parênteses) à esquerda do operador + é repetido **uma ou mais** vezes.

```
<digits> ::= (<digit>)+
```


- ? o símbolo (ou grupo de símbolos entre parênteses) à esquerda do operador ? é opcional (pode aparecer **zero ou uma** única vez).

```
<number> ::= <digits> ( "." <digits> )?
```


Linguagens de Programação

Estruturas Semânticas

- Significados de blocos sintáticos

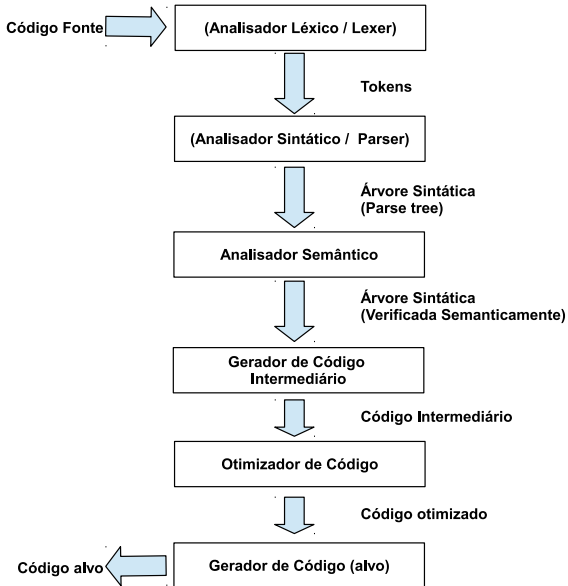


```
for (i = 1; i < 4; i++) {  
    //comandos  
}
```

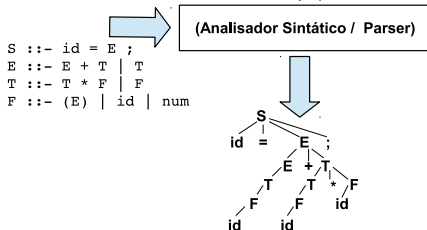
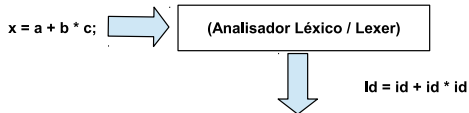


```
loop:  
    mov ax, 0  
    mov cx, 4  
    //comandos  
    inc ax  
    cmp cx, ax  
    jne loop
```

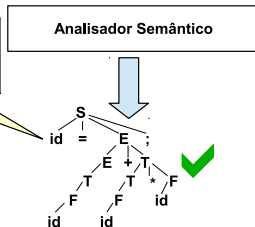
Linguagens de Programação



Linguagens de Programação



Não pode ser constante, função, ou palavra reservada e precisa ser compatível com o tipo do lado direito.



Analizador Léxico

Converte o “código fonte de entrada” em uma lista de tokens.

- Linguagem regular / expressão regular
- Utiliza autômatos finitos

Analizador Sintático

Converte a “lista de tokens” em uma árvore sintática.

- Linguagem livre de contexto
- Utiliza autômatos com pilha

Analizador Semântico

Analisa a “árvore sintática” semanticamente.

- Declaração de variáveis
- Escopo
- Verificação de tipos
- Conversões de tipos
- Declaração e uso de rótulos

Linguagens de Programação

Acesso a uma **tabela de símbolos** e **tratamento de erros** são realizados em todas as três fases de análise (léxica, sintática e semântica).

Tabela de símbolos

- Gerencia cada token (identificador), seu lexema e atributos, quando for o caso.

Tratamento de erros

- Informa erros em cada uma das fases. Exemplos:
 - Léxico: identificador mal formado (caracteres inválidos)
 - Sintático: encontrado **for**, esperado ;
 - Semântico: variável **x** não declarada

Atividades

- 1 Faça um resumo/tabela sobre alguns paradigmas de programação e cite as principais características de cada um deles. Cite também linguagens de cada um dos paradigmas.
- 2 Pesquise sobre outras linguagens de programação existentes e seus respectivos paradigmas.
- 3 O que é o paradigma de programação “orientado a agentes”? Pesquise sobre eles e cite algumas linguagens.
- 4 Leia o capítulo 1 do livro: AHO, A.V.; SETHI, R. ULLMAN, J.D. Compiladores - Princípios, Técnicas e Ferramentas, Ed. Addison Wesley 2008 / LTC, 1995. **(BU)**

Referências

Slides inspirados no material do prof. Joao Dovicchi.