

Programação Vetorial

Introdução

Prof. Edson Alves

Faculdade UnB Gama

Sumário

1. **Introdução**
2. **APL**
3. **Ambiente de Desenvolvimento**
4. **Conceitos elementares da APL**

Programação Vetorial

- ▶ A **programação vetorial** (*array programming*) é um paradigma de programação que permite a aplicação de operações em um conjunto de valores de uma só vez
- ▶ É usado predominantemente na programação científica ou em engenharias
- ▶ Linguagens que suportam a programação vetorial são denominadas linguagens **vetoriais** ou **multidimensionais**
- ▶ As primitivas dessas linguagens expressam concisamente ideias sobre manipulação de dados
- ▶ Não é incomum encontrar códigos de uma só linha em programação vetorial que equivalem a códigos de dezenas de linhas em outros paradigmas
- ▶ Exemplos de linguagens que suportam a programação vetorial: APL, J, MATLAB, Mathematica, Octave, R, etc

Principais características da programação vetorial

- ▶ Uma vez que as operações agem em coleções de objetos de uma só vez, é possível pensar e operar em dados sem uso de laços explícitos ou operações escalares
- ▶ Este paradigma estimula o pensamento dos dados em blocos de elementos correlatos e a exploração das propriedades destes elementos
- ▶ As funções são classificadas de acordo com o número de dimensões dos dados sob os quais elas agem
- ▶ Funções escalares (*rank* 0) agem em elementos de dimensão zero. Exemplo: adição nos inteiros
- ▶ Funções vectoriais agem em vetores (dados unidimensionais). Ex.: produto vetorial
- ▶ Funções matriciais, como multiplicação matricial, agem em matrizes (elementos bidimensionais, *rank* 2)

Kenneth E. Iverson

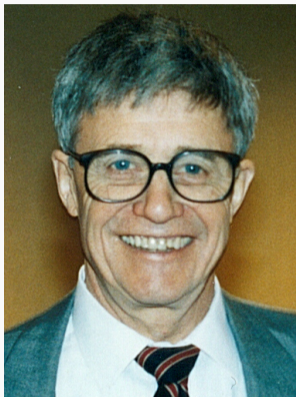


Figura: Kenneth Eugene Iverson (★1920 – †2004)

Iverson e a notação matemática

- ▶ Ao longo de sua carreira acadêmica, Iverson sempre esteve especialmente interessado na notação matemática
- ▶ Ele achava a notação matemática inconsistente e imprecisa
- ▶ Por exemplo, na matemática alguns operadores são pré-fixados, como o sinal de negativo ($-n$), e outros são pós-fixados, como no caso do fatorial ($n!$)
- ▶ No caso do valor absoluto, o operando fica cercado por dois símbolos ($|n|$)
- ▶ Algumas operações de fato nem tem símbolos associados, como a exponenciação, que anota o expoente como superescrito da base (a^b)
- ▶ Além disso, as diferentes regras de precedência da matemática levam ao uso de parêntesis e ambiguidades (a expressão a/bx significa $a/(bx)$ ou $(a/b)x$?)

Criação da APL

- ▶ Na percepção de Iverson, nem a notação matemática nem linguagens de programação como Fortran permitiam a expressão, publicação e discussão fluente de algoritmos
- ▶ Este pensamento o levou a desenvolver e propor uma nova notação em 1957, enquanto lecionava em Harvard
- ▶ Posteriormente, enquanto trabalhava na IBM e desenvolvia sua nova notação, ele e seus colaboradores a chamavam *Iverson's Better Math*
- ▶ A empresa não gostou do trocadilho e o nome da notação mudou para *A Programming Language* – APL
- ▶ Este nome foi usado oficialmente pela primeira vez em 1962, quando Iverson publicou o livro *A Programming Language*

APL – A notação e a linguagem

- ▶ A notação APL foi usada internamente na IBM na década de 60
- ▶ A nova notação proposta por Iverson permitiria a escrita de programas de computador claros e concisos
- ▶ Contudo, a proposta inicial, descrita no livro, não podia ser replicada ou inserida diretamente em um computador
- ▶ Por meio do apoio de colaboradores, como Adin Falkoff, Iverson elaborou uma nova APL que poderia ser usada em sessões interativas em um computador
- ▶ A linguagem APL é baseada na notação de Iverson, descrita no livro homônimo, e, embora diferente, mantém vários de seus conceitos fundamentais

Desenvolvimentos iniciais da APL

- ▶ Em 1962 foi feita a primeira tentativa de descrever um sistema computacional completo baseado em APL, motivada por uma discussão entre Falkoff e o Dr. William C. Carter
- ▶ No ano seguinte o Dr. Herbert Hellerman implementou parte da notação em um IBM 1620 e estudantes do ensino médio usaram esta implementação (PAT)
- ▶ Ainda em 1963 Falkoff, Iverson e Edward H. Sussenguth Jr. trabalharam juntos em uma implementação da APL, visando o desenvolvimento de programação e uso de computadores na educação
- ▶ No ano de 1965 Lawrence M. Breed e Philip S. Abrams implementaram uma parte da notação em FORTRAN IV (chamada IVSYS – *Iverson System*)
- ▶ Assim como no PAT, IVSYS ainda não usavam glifos para representar funções e operandos, mas palavras reservadas em inglês

Evolução da APL

- ▶ A evolução do APL passou, no ano de 1964, pelo desenvolvimento de hardwares capazes de inserir e imprimir os glifos
- ▶ A primeira tentativa de uma sessão interativa de APL foi feita por Larry Breed em 1966
- ▶ A IBM introduziu a APL no mercado em 1967, em um computador IBM 1130
- ▶ Na década de 70 a APL era utilizada por pesquisadores da IBM, na NASA e em instituições financeiras
- ▶ APL ganhou terreno nos *mainframes* entre as décadas de 60 e 80
- ▶ Em 1979, Iverson ganhou o Turing Award pelo seu trabalho na APL
- ▶ Décadas depois, Iverson também inventou a linguagem J, uma variante da APL que usa caracteres ASCII ao invés dos glifos

APL2

- ▶ Em 1980, Dr. Jim Brown liderou uma nova implementação da APL que permita *arrays* aninhados (isto é, um *array* pode conter outros *arrays*), denominada APL2
- ▶ Iverson não controlava mais o desenvolvimento da linguagem e deixou a IBM, se juntando a I. P. Sharp Associates para desenvolver a linguagem Sharp APL
- ▶ Mesmo hoje, a maioria das implementações de APL é compatível com APL2
- ▶ A primeira implementação de APL para microcomputadores data de 1973
- ▶ No início da década de 80 foi desenvolvido, pela Analogic Corporation, o “*The APL Machine*”, um computador capaz de processar *arrays* e projetado para ser programado em APL
- ▶ Foi um fracasso comercial, embora fosse o sistema APL mais rápido disponível até o momento
- ▶ A Microsoft chegou a cogitar uma implementação da APL, mas a ideia nunca se concretizou, embora hoje exista suporte para APL na plataforma .NET

Interpretadores APL

- ▶ Hoje em dia códigos APL são escritos predominantemente em plataforma Windows, com alguns casos em Linux, Unix e MacOS
- ▶ Há muito pouco registro de uso de APL em *mainframes*
- ▶ A APLNext oferece um interpretador APL avançado em Linux, Unix e Windows
- ▶ Dyalog APL é outro interpretador avançado, multiplataforma, que oferece uma série de extensões interessantes, como orientação a objetos e integração com a plataforma .NET
- ▶ A IBM também possui uma implementação de APL2 para IBM AIX, Linux, Sun Solaris e Windows
- ▶ NARS2000 é uma implementação open source da APL escrita por Bob Smith, primariamente para Windows
- ▶ A MicroAPL é a responsável pelo APLX, também multiplataforma, baseada no APL2 e com integração com .NET, Java, Ruby e R
- ▶ GNUAPL é a implementação da APL na suite GNU

Compiladores APL

- ▶ Em geral, programas APL são interpretados
- ▶ Na maioria dos casos, compiladores APL traduzem o código APL para linguagens de baixo nível, como C
- ▶ Compilação de código APL é um tema comum em conferências
- ▶ Alguns elementos de APL, como *arrays* aninhados, dificultam a compilação
- ▶ Como há diferenças significativas entre as versões e implementações de APL, a compilação seria uma alternativa ao processo aplicar as modificações nos códigos para funcionar em diferentes ambientes
- ▶ Um exemplo de compilador APL é o APEX, da Snake Island Research Inc, que traduz código APL para código SAC, uma linguagem funcional baseada em *arrays*

Dyalog APL

- ▶ A empresa Dyalog disponibiliza um ambiente de desenvolvimento baseado em APL, livre para uso não comercial
- ▶ A versão mais recente é a 19.0
- ▶ Há implementações disponíveis para Windows, MacOS, Linux e Raspberry Pi
- ▶ O download pode ser feito na página oficial: dyalog.com
- ▶ Em ambiente Linux, uma vez instalado, pode-se iniciar uma nova sessão por meio do comando

```
$ dyalog
```

- ▶ Para sair do ambiente interativo, use o comando `)off`

Teclado APL

Para inserir os glifos APL é preciso ou um teclado especializado ou a instalação de um *layout* compatível.

~ ◇	!⋮ 1"	@▽ 2 ⁻	#▽ 3<	\$Δ 4≤	%ϕ 5=	^Q 6≥	&Θ 7>	*⊙ 8≠	(▽ 9v)^ 0λ	_ -x	+÷ =÷	BACKSP	
TAB		Q q?	W⋮ w	E∈ e∈	R rρ	T~ t~	Y¥ y↑	U u↓	I⋮ i⋮	OÖ oö	P* p*	{⋮ [←	}θ]→	← \\←
(CAPS LOCK)		Aα aα	S s	D d	F▽ f_	G g▽	HΔ hΔ	J∘ j°	K⊠ k'	L⋮ l⋮	:≡ ;⊥	"≠ '⊥	RETURN	
SHIFT			Z z<	Xχ x>	C¢ cη	V vυ	B£ b⊥	N nτ	M m	<τ ,⊘	>χ .	?⋮ /≠	SHIFT	

Figura: *Layout* de teclado GNU APL. Fonte: [Bug-apl](#)

Configuração de Layout no Ubuntu 20.04

1. Nas configurações do sistema (*Settings*), escolha a opção de região e linguagem (*Region & Language*)
2. Adicione um segundo *layout* qualquer (por exemplo, o *layout* Braille) usando o símbolo + na lista de entradas (*Input Sources*)

3. Rode, no seu terminal, o comando

```
$ setxkbmap -layout br,apl -variant ,dyalog -option grp:switch
```

4. Para tornar esta configuração permanente, adicione esta linha no arquivo `.bashrc`
5. Além do *layout*, é recomendada a instalação da fonte APL 385 para melhor visualização dos glifos

Inserção de glifos APL


- ▶ Uma vez disponível o *layout* APL, os glifos podem ser inseridos por meio de uma combinação de teclas
- ▶ A tecla APL deve ser combinada com uma outra tecla, ou com a tecla `Shift` e outra tecla
- ▶ No *layout* proposto, a tecla APL corresponde a tecla `Alt Gr`
- ▶ Por exemplo, os glifos ρ e Ψ podem ser inserido por meio das combinações `APL+p` e `APL+Shift+3`, respectivamente
- ▶ Uma maneira alternativa é inserir os códigos unicode de cada caractere diretamente em seu editor
- ▶ Este método dispensa a configuração do *layout*, porém demanda a memorização dos códigos e do uso de mais teclas por caractere

Encerrando a sessão do Dyalog

- ▶ Uma vez instalado o *layout*, é possível encerrar uma sessão do Dyalog por meio de uma função do próprio APL
- ▶ Basta utilizar a função de sistema OFF
- ▶ Em APL, as funções de sistema tem seus nomes prefixados pelo símbolo ⎕ (quad)
- ▶ Esta opção demanda a inserção de um glifo para acessar a função de sistema, tarefa complicada sem o *layout* previamente configurado
- ▶ Para invocar a função de sistema OFF em uma sessão ativa do Dyalog, use o commando:

⎕OFF

Novo símbolo

Símbolo	Aridade	Descrição
 (<i>quad</i>)	monádico	Prefixo das funções de sistema

Unicode	TAB	APL
U+2395	[] <tab>	APL + l

TryAPL

- ▶ A Dialog disponibiliza um ambiente online chamado TryAPL para que deseje conhecer a linguagem
- ▶ Ela disponibiliza um modo de inserção baseado em uma combinação intuitiva de dois caracteres ASCII, seguidos de um TAB



Características da APL

- ▶ APL é uma linguagem dinamicamente tipada, interpretada e interativa
- ▶ Foi fortemente influenciada pela notação matemática
- ▶ Também influenciou as linguagens J, K, Mathematica, MATLAB, Nial, PPL, Q; as planilhas eletrônicas, a programação funcional e pacotes matemáticos computacionais
- ▶ Ela suporta predominantemente a programação vetorial, de modo que a estrutura de dados básica para armazenamento de dados é o *array*
- ▶ APL foca na solução do problema, enfatizando a expressão de algoritmos que independem da arquitetura ou do sistema operacional
- ▶ Ela promove o *problem solving* em um maior nível de abstração
- ▶ A generalidade de APL provém da simplicidade e uniformidade de suas regras

Características da APL

- ▶ APL automatiza os aspectos irrelevantes da programação, ampliando a produtividade
- ▶ Outro aspecto importante é que APL trata os números e suas conversões internamente, tornando desnecessário o conhecimento de suas representações, limites, etc
- ▶ Dada a natureza da APL, alguns a consideram uma linguagem “*write-only*”
- ▶ À primeira vista a leitura de um código APL remete à decodificação de hieroglifos egípcios
- ▶ A concisão e abrangência das funções de APL permitem escrever o Jogo da Vida de Conway em uma única linha!

```
life ← {↑1 ω∇.∧3 4=+/,~1 0 1◊.⊖~1 0 1◊.ϕ<ω}
```

Glifos

- ▶ Códigos APL utilizam glifos (símbolos), alguns oriundos do alfabeto grego, ao invés dos caracteres ASCII tradicionalmente usados em outras linguagens
- ▶ Alguns destes glifos são oriundos da própria matemática. Por exemplo: $+$, $-$, \times , \div , $!$, $<$, \leq , $=$, \geq , $>$, \neq , \vee , \wedge , \in
- ▶ Linguagens convencionais substituem alguns destes símbolos por outros símbolos ASCII
- ▶ Por exemplo, C/C++ utiliza. $*$ para multiplicação e $/$ para divisão
- ▶ APL denomina **primitivas** as características intrínsecas da linguagem, as quais são representadas por um ou mais símbolos (glifos)
- ▶ A maioria das primitivas são funções e operadores

Ordem de avaliação das expressões

- ▶ Em APL há apenas uma única e simples regra de precedência: o argumento à direita de uma função é o resultado de toda expressão à sua direita
- ▶ Esta ordem de avaliação parece estranha à princípio, mas traz vantagens
- ▶ Na ordem matemática, o algoritmo de Horner para computar o valor do polinômio $p(x) = a + bx + cx^2 + dx^3$ em y seria notado como

$$p(y) = a + y \times (b + y \times (c + y \times d))$$

$$a + y \times b + y \times c + d \times y$$

- ▶ Outro exemplo, em APL $2 \times 3 + 5$ resulta em 16, e não 11
- ▶ Esta escolha reduz substancialmente a necessidade de parêntesis

Funções, *arrays* e operadores

- ▶ APL distingue entre funções e operadores
- ▶ As funções recebem *arrays* retangulares como argumentos e retornam *arrays*
- ▶ *Arrays* retangulares tem zero ou mais dimensões, não necessariamente de mesmo tamanho
- ▶ Um vetor é um *array* de dimensão 1, uma matriz um *array* de dimensão 2
- ▶ Operadores permitem construir funções que são variantes de outras funções
- ▶ Eles remetem à funções de alta ordem e recebe funções ou *arrays* como argumentos e derivam funções relacionadas
- ▶ Por exemplo, a função `sum` pode ser derivada a partir o operador `/` (redução) e da função `+` (adição)

`sum` ← `+/`

Programas em APL

- ▶ Em uma sessão interativa, um ambiente APL é denominado um **workspace**, onde o usuário pode inserir e manipular dados sem definir um programa
- ▶ Programas em APL são cadeias de funções monádicas ou diádicas em conjunto com operadores e *arrays*
- ▶ Exemplo de programa APL, que ilustra a expressividade e concisão de APL: a função P abaixo retorna 1 se o argumento é um palíndromo, 0 caso contrário

$P \leftarrow \{ \wedge / \omega = \phi \omega \}$

- ▶ **Significado:** é verdade que, para todos os caracteres ($\wedge /$) do argumento ω , eles são iguais ($=$) ao caractere correspondente do reverso ($\phi \omega$) deste argumento?

Referências

1. APL Wiki. [Quad name](#), acesso em 23/09/2021.
2. **BROCKLEBANK**, Daniel. *APL – The Language*, John Hopkins APL Technical Digest, vol. 5, number 3, 1984.
3. Dyalog. [Dyalog APL Version 12.0](#), acesso em 23/09/2021.
4. Dyalog. [Try APL](#), acesso em 23/09/2021.
5. **IVERSON**, Kenneth E. *A Programming Language*, John Wiley and Sons, 1962.
6. Stack Overflow. [Add APL Keyboard Layout On Linux 20.04](#), acesso em 23/09/2021.
7. **ULMANN**, Bernd. *APL - One of the Greatest Programming Languages Ever*, Vintage Computer Festival Europe 2007.
8. Xah Lee. [Unicode APL Symbols](#), acesso em 23/09/2021.
9. Wikipédia. [APL \(programming language\)](#), acesso em 24/09/2021.
10. Wikipédia. [Array programming](#), acesso em 24/09/2021.