MAIÊUTIC: UMA LINGUAGEM DE PROGRAMAÇÃO INSPIRADA NO MÉTODO SOCRÁTICO

Vinícius Rodrigues de Freitas

1. Introdução

A linguagem Maiêutic nasce no contexto da disciplina de Lógica da Computação como uma proposta de unir filosofia e programação. Em vez de focar apenas em algoritmos tradicionais, a ideia é criar uma linguagem capaz de expressar roteiros de investigação — com perguntas, respostas, testes de consistência e conclusões — inspirados na prática de diálogo de Sócrates.

Este documento apresenta a motivação da linguagem, suas principais características, alguns exemplos de uso e um resumo de como ela é implementada e executada por meio de um compilador e de uma máquina virtual chamada SocraticVM.

2. Motivação

A tradição socrática vê o conhecimento não como algo simplesmente transmitido, mas como algo descoberto através de perguntas. Sócrates não oferecia respostas prontas: ele conduzia seus interlocutores a examinar suas crenças, encontrar contradições e, a partir disso, construir entendimentos mais sólidos.

A linguagem Maiêutic tenta levar esse espírito para o mundo da computação. Em vez de apenas descrever passos para calcular um resultado, o programador escreve um roteiro de diálogo, que faz perguntas ao usuário, registra suas respostas em variáveis, testa essas respostas logicamente, registra tentativas descartadas e, ao final, produz uma síntese ou conclusão.

A linguagem foi pensada para ser legível em diferentes contextos: em um terminal, impressa em papel como pseudocódigo ou mesmo usada como roteiro mental de autoquestionamento.

3. Filosofia: a maiêutica de Sócrates

O nome da linguagem vem do termo grego “maiêutica”, que significa “arte de partejar”. Para Sócrates, o professor não coloca conhecimento na mente do aluno; ele ajuda a trazer à luz as ideias que já estão, de algum modo, latentes.

Um programa escrito em Maiêutic procura simular esse processo. Em vez de dizer ao usuário que felicidade é algo fixo, o programa pergunta como ele a define, testa essa definição, busca contraexemplos, aponta fragilidades e convida o usuário a reformular sua resposta. A cada iteração, a definição vai sendo refinada, até que se chegue a algo mais robusto.

Assim, a linguagem não é apenas um exercício técnico: ela é também uma forma de explorar como processos filosóficos podem ser formalizados em termos de lógica e controle de fluxo.

4. Visão geral da linguagem Maiêutic

A Maiêutic é uma linguagem de alto nível, com sintaxe própria, construída em torno de alguns conceitos fundamentais: perguntas ao usuário, captura de respostas, mensagens intermediárias, conclusões e estruturas clássicas de controle de fluxo, como condicionais e laços de repetição.

Um programa é uma sequência de comandos organizados por indentação, em estilo similar ao da linguagem Python. Cada execução da linguagem é, na prática, uma sessão de diálogo entre o sistema e o usuário.

5. Elementos básicos da linguagem

5.1 Variáveis e memória

Na Maiêutic, as variáveis representam o estado do conhecimento do programa: definições, hipóteses, listas de erros, contadores e outros dados. Todas as variáveis são prefixadas com o símbolo arroba (@).

```
@conceito  := "Felicidade"
@definicao := ""
@incerteza := Verdadeiro
@historico_erros := []
```

Atribuições são feitas com o operador de definição. Listas podem crescer dinamicamente com o operador de adição ao final, que adiciona um elemento ao fim da lista.

```
@historico_erros << @definicao
```

O acesso a um elemento específico de uma lista é feito por índice, usando notação de colchetes.

```
@primeiro := @historico_erros[0]
```

5.2 Perguntas, respostas e registros

Três operadores formam a interface socrática da linguagem: o operador de pergunta inicia uma questão ao usuário; o operador de entrada lê uma resposta e a armazena em uma variável; e o operador de log exibe uma mensagem intermediária sem esperar resposta.

```
? "Como você define " + @conceito + " em poucas palavras?"
> @definicao
>> "Analisando a proposição: '" + @definicao + "'"
```

O código alterna entre perguntar, registrar e comentar o que está acontecendo, criando um fluxo de interação contínua.

5.3 Conclusões

A conclusão é um tipo especial de saída, marcada por um operador específico. Ela costuma representar o resultado final do roteiro de investigação.

```
! "Sua definição madura de Felicidade é: " + @definicao
```

Embora tecnicamente seja apenas uma forma diferente de exibir uma mensagem, semanticamente ela carrega o papel de síntese final do processo dialético que o programa conduziu.

5.4 Controle de fluxo: Se, Senao e Enquanto

A linguagem suporta condicionais e laços de repetição com palavras-chave em português. Um bloco condicional é escrito com as palavras Se e Senao, enquanto os laços usam a palavra Enquanto.

```
-> Se @existe_contradicao == "Sim":
    >> "Então essa definição é insuficiente."
-> Senao:
    >> "Não encontramos contradições por enquanto."

Enquanto @incerteza == Verdadeiro:
    ? "Consegue pensar em um contraexemplo?"
    > @existe_contradicao
```

Essas estruturas permitem que o programa retorne a perguntas anteriores, repita testes e explore diferentes ramos de raciocínio conforme as respostas do usuário.

5.5 Expressões, tipos e listas

As expressões da linguagem incluem operadores aritméticos, relacionais e lógicos, além de literais numéricos, textuais e booleanos, listas com acesso por índice e uma função de tamanho que funciona para listas e cadeias de caracteres.

Os principais tipos de dado são números, cadeias de caracteres, valores booleanos e listas. O design favorece operações simples, porém suficientes para expressar tanto raciocínios conceituais quanto pequenos algoritmos numéricos.

6. Exemplos de programas

6.1 Investigando a felicidade

Um dos exemplos clássicos da linguagem é um roteiro que investiga o conceito de felicidade. O programa pede uma definição inicial, testa sua consistência e convida o usuário a refiná-la.

```
@conceito  := "Felicidade"
@definicao := ""
@incerteza := Verdadeiro
@historico_erros := []

>> "Iniciando investigação sobre: " + @conceito

? "Para começar, como você define " + @conceito + " em poucas palavras?"
> @definicao

Enquanto @incerteza == Verdadeiro:

    >> "Analisando a proposição: '" + @definicao + "'"

    ? "Consegue imaginar alguém que possui isso (" + @definicao + ") mas ainda e
    > @existe_contradicao

    -> Se @existe_contradicao == "Sim":
        @historico_erros << @definicao
        ? "O que falta a essa pessoa, mesmo tendo " + @definicao + "?"
        > @elemento_faltante
        ? "Tente uma nova definição que inclua " + @elemento_faltante + ":"
        > @definicao

    -> Senao:
        @incerteza := Falso

>> "Você abandonou " + tamanho_de(@historico_erros) + " definições superficiais
! "Sua definição madura de Felicidade é: " + @definicao
```

O resultado é uma sessão de diálogo em que o próprio usuário desmonta definições frágeis e constrói, passo a passo, uma concepção mais cuidadosa de felicidade.

6.2 Verificador de números primos

Para mostrar que a linguagem também suporta algoritmos mais tradicionais, há um exemplo de verificador de números primos, que combina interação com o usuário e cálculo aritmético.

```
@numero := 0
@divisor := 2
```

```
@eh_primo := Verdadeiro

? "Digite um número inteiro positivo para verificar primariedade:"
> @numero

-> Se @numero <= 1:
    @eh_primo := Falso
    >> "Números menores ou iguais a 1 não são primos."

-> Senao:
    Enquanto (@divisor * @divisor) <= @numero:
        -> Se (@numero % @divisor) == 0:
            @eh_primo := Falso
            @divisor := @numero
        @divisor := @divisor + 1

-> Se @eh_primo == Verdadeiro:
    ! "O número " + @numero + " É PRIMO."
-> Senao:
    ! "O número " + @numero + " NÃO É PRIMO.
```

Esse programa dialoga com o usuário apenas na entrada inicial, mas o corpo do algoritmo é um teste clássico de primalidade. Assim, a Maiêutic pode ser vista tanto como linguagem de diálogo quanto como linguagem de programação geral.

7. Implementação e execução

Embora o foco deste texto seja a linguagem em si, é útil ter uma visão de como ela é executada na prática. O projeto contém um compilador em C++, construído com Flex e Bison, que lê o código-fonte, constrói uma árvore sintática abstrata e gera um arquivo de assembly específico da linguagem.

Esse assembly é então executado por uma máquina virtual em Python chamada SocraticVM, que interpreta as instruções em uma máquina de pilha. A máquina implementa operações aritméticas, lógicas e de manipulação de listas, bem como instruções específicas para interação socrática, como pergunta, leitura de entrada e emissão de conclusões.

O fluxo completo de uso consiste em escrever um arquivo Maiêutic, compilá-lo com o compilador maieutic e executar o arquivo resultante na SocraticVM, que conduz a interação com o usuário.

8. Curiosidades e decisões de projeto

Algumas escolhas ajudam a dar identidade própria à linguagem. As palavras-chave são em português, o que reforça o caráter didático e aproxima o código de um roteiro em língua

natural. A estrutura de blocos usa indentação, favorecendo a leitura como se fosse pseudocódigo.

O sistema de entrada tenta converter respostas textuais automaticamente: números são lidos como valores numéricos; respostas como Sim ou Verdadeiro e Nao ou Falso são interpretadas como valores booleanos; e demais textos são tratados como cadeias de caracteres.

Listas permitem registrar históricos de pensamento, como definições já descartadas ou exemplos fornecidos pelo usuário, o que combina bem com a ideia de um diário de investigação.

9. Possibilidades futuras

A linguagem Maiêutic abre diversas possibilidades para exploração. Entre elas, a criação de uma biblioteca de roteiros educativos em áreas como lógica, ética, matemática e filosofia da ciência, a extensão da linguagem com funções definidas pelo usuário, módulos e tipos adicionais e o desenvolvimento de novos sensores na máquina virtual.

Outra linha de trabalho é o desenvolvimento de interfaces gráficas ou aplicações web que transformem programas Maiêutic em assistentes socráticos interativos, ampliando o alcance da linguagem para além do ambiente de linha de comando.

10. Conclusão

A linguagem Maiêutic é uma experiência de aproximar a programação do ato de pensar junto. Em vez de se limitar a cálculos e operações mecânicas, ela propõe que um programa possa ser um parceiro de diálogo: alguém que pergunta, registra, testa, insiste e, no fim, ajuda o usuário a articular melhor aquilo que pensa.

Do ponto de vista da lógica da computação, o projeto mostra como conceitos clássicos — variáveis, expressões, controle de fluxo, compiladores e máquinas virtuais — podem ser reorganizados para servir a um propósito distinto: não apenas resolver problemas, mas estruturar o processo de investigação que leva às respostas.

Em resumo, a Maiêutic é um convite a programar perguntas para descobrir, junto com o usuário, quais respostas realmente fazem sentido.