**MINI COMPILADOR**

Este relatório orienta o utilizador na usabilidade desta que é considerada a segunda ferramenta fundamental para a execução de um programa de computadores. A linguagem a ser analisada é Java. Para a construção do presente “mini compilador”, foi necessário passar por três etapas fundamentais que envolvem a construção de um compilador, respectivamente, análise léxica, análise sintática e análise semântica .

Antes de mais, a seguir, defino os aspectos fundamentais deste ramo.

**DEFINIÇÕES:**

**Compilador**

Definição de Compilador, segundo o livro "Compiladores: Princípios e Práticas":

"Um **compilador** é um programa que traduz um programa escrito em uma linguagem de programação de alto nível para uma forma equivalente em uma linguagem de baixo nível, como linguagem de máquina ou código intermediário, que pode ser executado diretamente por um computador."

**Análise léxica** ou **scanner:** é a primeira fase da implementação de um com­pila­dor. O objetivo da análise léxica é dividir o texto de entrada, entre­gando uma sequên­cia de comentários e símbolos elementares chamados lexemas. Cada lexe­ma é um par consistindo de um nome de token e um valor de atributo opcional. [...]". (Aho, Sethi e Ullman - Compilers: Principles, Techniques, and Tools “Dragon Book”).

O **Análise sintática** ou **parse:** é a segunda fase do Compilador, responsável por validar a sintaxe de um subconjunto de código fonte da linguagem em estudo (Java). Este manual tem como objectivo fornecer informações detalhadas sobre o desenvolvimento e uso do Analisador Sintático, abordando aspectos como arquitetura e funcionamento.

**Análise semântica:** é a fase de um compilador em que é verificado se as construções sintáticas do programa fonte estão de acordo com as regras semânticas da linguagem. Nessa fase, informações semânticas são coletadas e verificadas para garantir a corretude e coerência do programa, como a verificação de tipos, escopo de variáveis, compatibilidade de operações e outras regras específicas da linguagem.

**ARQUITETURA DO MINI COMPILADOR**

O mini compilador está composto pelos seus três componentes fundamentais: o Analisador Léxico, Analisador Sintático, este combinado com o Analisador Semântico. O Analisador Léxico é responsável por transformar o fluxo de entrada em tokens, enquanto o Analisador Sintático utiliza esses tokens para validar a estrutura gramatical do código fonte de acordo a gramática da linguagem em estudo (Java), e o Semântico analisa se as construções sintáticas do programa fonte estão de acordo com as regras semânticas da linguagem.

O presente Manual instrui o programador na utilização e compreensão dos aspectos técnicos deste “mini compilador”.

**ESPECIFICAÇÕES DO ANALISADOR LÉXICO**

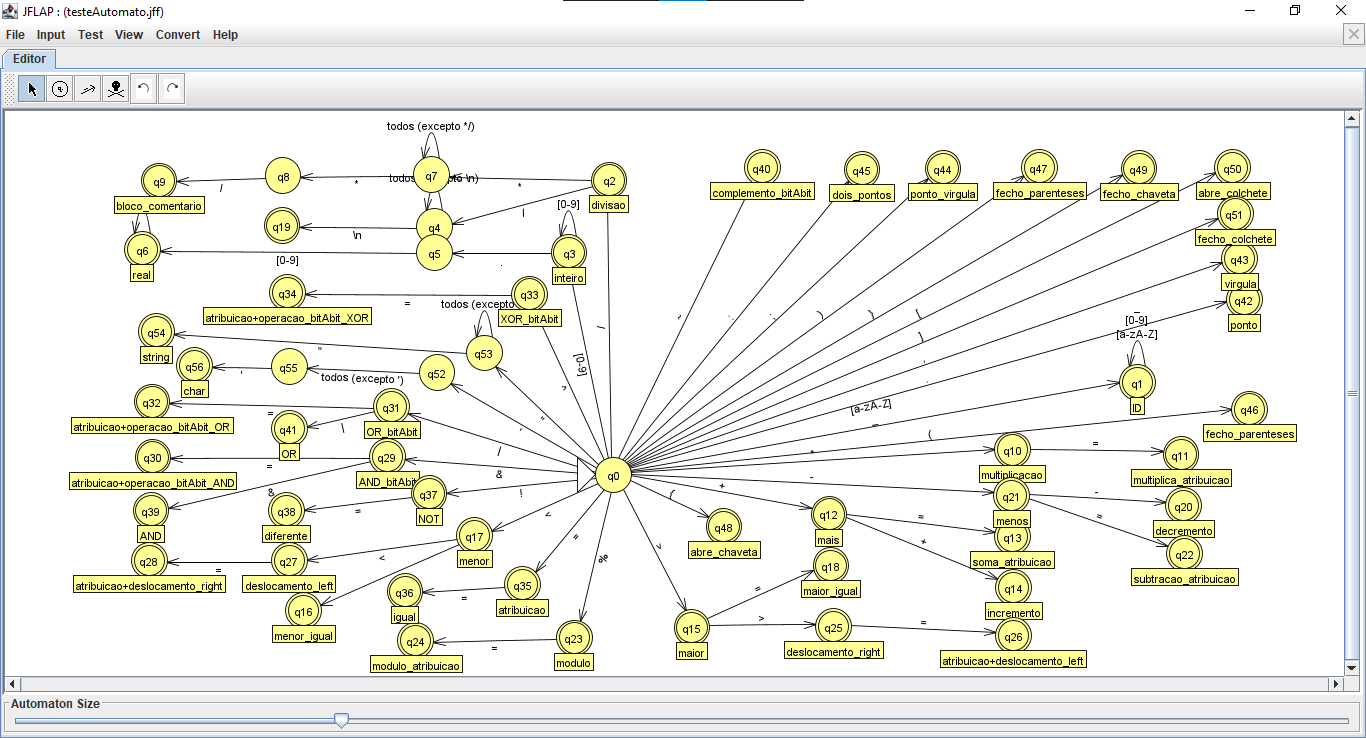
**LISTA DE TOKENS UTILIZADOS**

Foram validados os seguintes tokens no Analisador Léxico:

|  |
| --- |
| *TK\_ID* |
| *TK\_COMENTARIO* |
| *TK\_BLOCO\_COMENTARIO* |
| *TK\_OP\_DIVISAO* |
| *TK\_OP\_DIVISAO\_ATRIBUICAO* |
| *TK\_NUM\_INTEIRO* |
| *TK\_NUM\_REAL* |
| *TK\_OP\_MAIS* |
| *TK\_OP\_INCREMENTO* |
| *TK\_OP\_SOMA\_ATRIBUICAO* |
| *TK\_OP\_MENOS* |
| *TK\_OP\_DECREMENTO* |
| *TK\_OP\_SUBTRACAO\_ATRIBUICAO* |
| *TK\_OP\_MULTIPLICACAO* |
| *TK\_OP\_MULTIPLICACAO\_ATRIBUICAO* |
| *TK\_OP\_ATRIBUICAO* |
| *TK\_OP\_IGUAL* |
| *TK\_OP\_XOR\_BIT\_A\_BIT* |
| *TK\_OP\_XOR\_BIT\_A\_BIT\_ATRIBUICAO* |
| *TK\_OP\_OR\_BIT\_A\_BIT* |
| *TK\_OP\_OR\_BIT\_A\_BIT\_ATRIBUICAO* |
| *TK\_OP\_OR* |
| *TK\_OP\_AND\_BIT\_A\_BIT* |
| *TK\_OP\_AND\_BIT\_A\_BIT\_ATRIBUICAO* |
| *TK\_OP\_AND* |
| *TK\_OP\_DIFERENTE* |
| *TK\_OP\_NOT* |
| *TK\_OP\_MODULO* |
| *TK\_OP\_MODULO\_ATRIBUICAO* |
| *TK\_OP\_MENOR* |
| *TK\_OP\_MENOR\_IGUAL* |
| *TK\_OP\_DESLOCAMENTO\_LEFT* |
| *TK\_OP\_DESLOCAMENTO\_LEFT\_ATRIBUICAO* |
| *TK\_OP\_MAIOR* |
| *TK\_OP\_MAIOR\_IGUAL* |
| *TK\_OP\_DESLOCAMENTO\_RIGHT* |
| *TK\_OP\_DESLOCAMENTO\_RIGHT\_ATRIBUICAO* |
| *TK\_OP\_COMPLEMENTO\_BIT\_A\_BIT* |
| *TK\_VALOR\_STRING* |
| *TK\_VALOR\_CHAR* |
| *TK\_ABRE\_PARENTESES* |
| *TK\_FECHA\_PARENTESES* |
| *TK\_ABRE\_CHAVETA* |
| *TK\_FECHA\_CHAVETA* |
| *TK\_ABRE\_COLCHETES* |
| *TK\_FECHA\_COLCHETES* |
| *TK\_DOIS\_PONTOS* |
| *TK\_PONTO\_VIRGULA* |
| *TK\_VIRGULA* |
| *TK\_PONTO* |
| *TK\_KEYWORD* |
| *EOF* |

**AUTÓMATO**

Durante a implementação do Analisador Léxico, utilizou-se o autómato abaixo como base. Foi feita uma adaptação do autómato na implementação, pois este autómato deixa implícito os estados “outros” que fazem parar uma transições. Foram considerados como estados finais todos aqueles que neles a expressão é válida, ao invés de se aplicar o estado “outro”.



**EXPRESSÃO REGULAR**

O autómato acima aceita/valida a seguinte expressão regular (gerada pelo JFLAP):

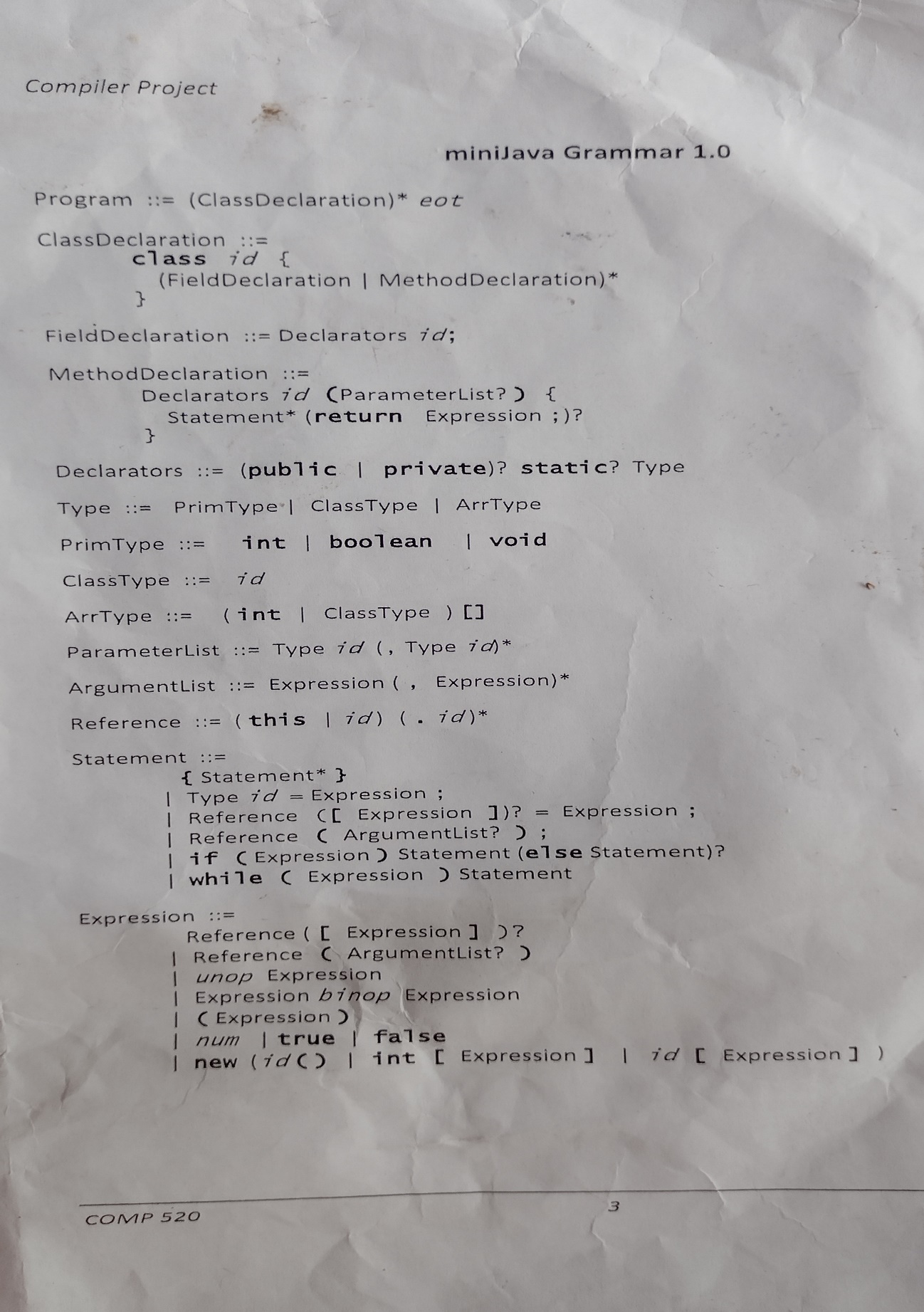
*(\_+[a-zA-Z])([a-zA-Z]+[0-9]+\_)\*+/+[0-9]([0-9])\*+[0-9]([0-9])\*.[0-9]([0-9])\*+/\*(todos (excepto \*/))\*\*/+\*+\*=+++(+)=+(+)(+)+>+<(λ+=)+>=+//(todos (excepto \n))\*\n+-(λ+-)+-=+%+%=+>>+>>=+<<+<<=+&+&=+|+|=+^+^=+=+==+!+!=+&&+~+||+.+,+;+:+(+)+{+}+[+]+"(todos (excepto "))\*"+'todos (excepto ')'*

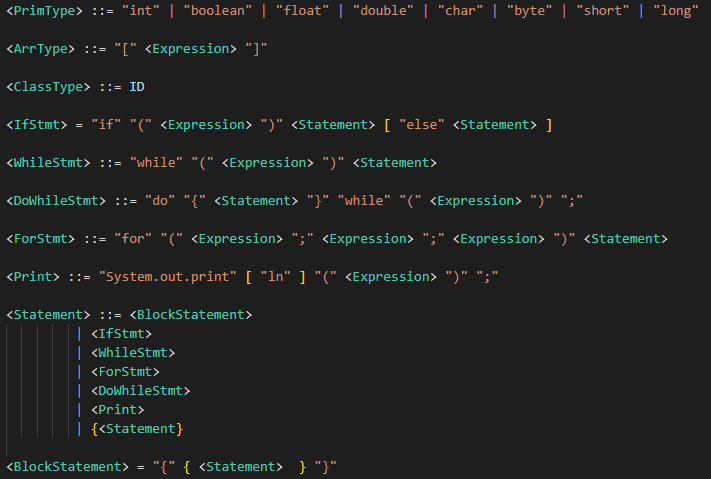
**ESPECIFICAÇÕES DO ANALISADOR SINTÁTICO**

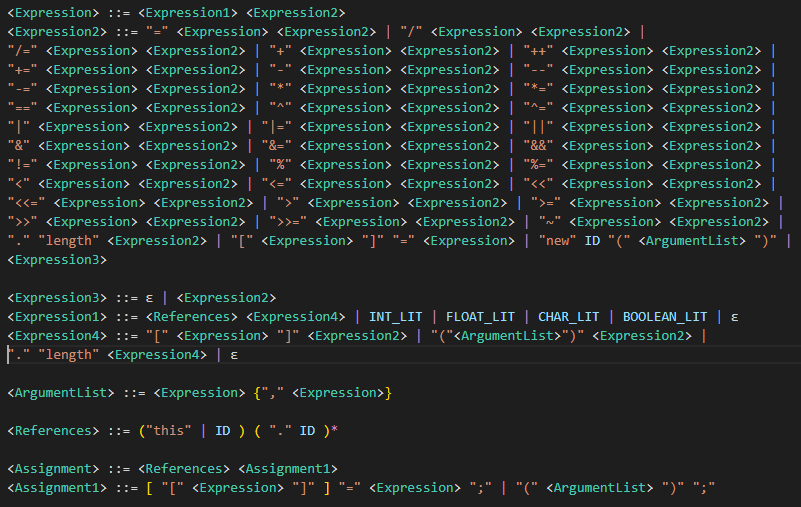
**GRAMÁTICA DA LINGUAGEM JAVA**

O Analisador Sintático suporta um subconjunto da linguagem Java e requer uma descrição detalhada da sua gramática. A gramática inclui regras gramaticais, símbolos terminais e não-terminais que definem a estrutura da linguagem Java. É essencial compreender e especificar corretamente a gramática para que o Analisador Sintático possa validar a sintaxe corretamente.

**GRAMÁTICA BASE (NÃO TRABALHADA)**

****

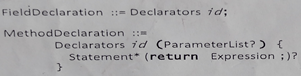
**GRAMÁTICA ORGANIZADA (TRABALHADA)**



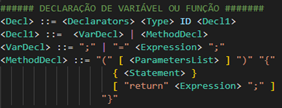
**ALGUMAS AMBIGUIDADES TRATADAS**

Declaração de variáveis e funções: <FieldDeclaration> e <MethodDeclaration>

Antes:



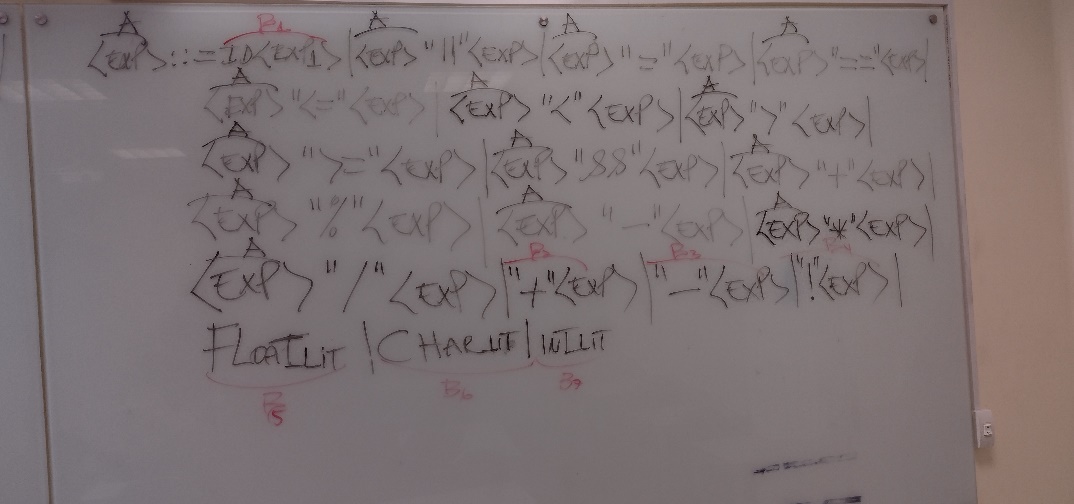
Depois:



**ALGUMAS RECURSIVIDADES À ESQUERDA TRATADAS**

Na validação de Expression (expressão): <Expression >

Antes:



Depois



**IMPLEMENTAÇÃO**

O Analisador Sintático foi implementado na linguagem C e é composto por vários arquivos. O arquivo "analex.c" contém o código do Analisador Léxico, enquanto o arquivo "parser.c" contém a lógica do Analisador Sintático. Há também arquivos de cabeçalho para os tokens, para o TDA léxico (lista que armazena informações dos lexemas) e um arquivo implementando o TDA léxico. A interação entre o Analisador Léxico e o Analisador Sintático é fundamental para o funcionamento adequado do sistema.

**FUNCIONAMENTO DO ANALISADOR SINTÁTICO**

O funcionamento do Analisador Sintático ocorre em etapas. A função principal é a "parser", que é chamada inicialmente pela função "main". A função "parser" é responsável por iniciar a análise sintática, começando pelo não-terminal "program", que é o ponto de partida da gramática. O Analisador Sintático recebe os lexemas/tokens a partir do Análisador Léxico e valida sua sintaxe. Os resultados da análise são exibidos na linha de comando.

**TRATAMENTO DE ERROS**

O Analisador Sintático deve ser capaz de identificar e tratar erros léxicos e sintáticos. Erros léxicos são aqueles relacionados aos tokens inválidos, enquanto erros sintáticos estão relacionados à estrutura gramatical incorreta. É importante implementar estratégias de recuperação de erros para lidar com essas situações e fornecer mensagens de erro claras e informativas. Os erros foram armazenados em listas ligadas para facilitar o tratamento posterior.

**DECISÕES DE IMPLEMENTAÇÃO**

Durante a implementação do Analisador Sintático, algumas decisões são cruciais para o seu funcionamento adequado. Uma das principais dificuldades esteve na decisão de quando ler o próximo token e quando não ler, a fim de evitar comportamentos indesejados do Parser. É fundamental ter cuidado ao implementar essa lógica e garantir que o Parser funcione corretamente.

**REFERÊNCIAS QUE PODEM GUIAR O PROGRAMADOR**

Livros:

* **Compiladores - Princípios e Práticas** (Kenneth C. Louden);
* **Compiladores - Princípios, Técnicas e Ferramentas** (Alfred V. Aho, Ravi Sethi e Jeffrey D. Ullman);
* **Engineering a Compiler** (Keith D. Cooper e Linda Torczon).