

Universidade de São Paulo  
Instituto de Ciências Matemáticas e de Computação  
Linguagens de Programação e Compiladores

# Analizador Léxico

Aluno: Bruno Henrique Rasteiro N°USP: 9292910

Aluno: Kairo Bonicenha N°USP: 9019790

Aluno: Carlos André Martins Neves N°USP: 8955195

Aluno: Marly da Cruz Cláudio N°USP: 8936885

Aluno: Tobias Mesquista Silva da Veiga N°USP: 5268356

Professor: Diego Raphael Amancio

São Carlos  
2019

# 1 Compilação e execução do analisador léxico

O analisador foi implementado utilizando o programa *flex*, para facilitar sua compilação foi criado um arquivo *MakeFile* com os seguintes comandos:

**\$ make** - Compila o programa e gera o executável *lex* no diretório raiz

**\$ make test** - Compila e executa os casos de teste da pasta */tests*

Para executar o programa basta chamar o executável passando o arquivo de entrada desejado como *stdin*. A saída da análise será tanto impressa tanto no *stdout*, quanto no arquivo *output.txt* no diretório raiz.

## 2 Decisões de projeto

A primeira decisão de projeto tomada foi na identificação de símbolos e palavras reservadas, onde optamos por retornar a própria palavra reservada ou o próprio símbolo identificado. Acreditamos que dessa forma a saída do analisador fica mais simples e legível.

Outra decisão de projeto foi a implementação de um automato para identificar as palavras reservadas, esse tópico é discutido na seção 2.1.

As demais decisões vieram nos casos onde não era trivial o reconhecimento de um *token* ou quando um erro deveria ser relatado, nesses casos as decisões foram tomadas com base no comportamento do compilador GCC. As subseções 2.2, 2.3 e 2.4 exemplificam as dúvidas e as decisões tomadas.

### 2.1 Tabela de palavras reservadas

Utilizamos o algoritmo de um autômato para diferenciar entre identificadores e palavras reservadas. A implementação consiste apenas de regras condicionais. Além do autômato, tentamos a implementação de uma hashtable. Para comparar as implementações foi realizado um teste de repetir 1 milhão de vezes a consulta de 32 tokens (16 reservados e 16 não reservados). O autômato foi mais eficiente executando em 0.33s contra os 0.90s da hashtable. Os arquivos com as implementações para teste encontram-se no nosso projeto em *src/time\_tests/*.

### 2.2 Símbolo não pertencente ao alfabeto

Sempre que o analisador se depara com um símbolo/carácter que não pertence ao alfabeto da linguagem, é emitido uma mensagem de erro informando ao usuário qual o carácter inválido e a linha do ocorrido. A Tabela 1 mostra alguns exemplos de código fonte (Entrada) e de saída do analisador (Saída) para exemplificar como ele se comporta nesse contexto.

Tabela 1: Caso de teste *input0.txt* para tratamento de erro ao encontrar um símbolo que não pertence ao alfabeto.

Linha	Entrada	Saída
1	asd	asd - id
2	@sd sd - id	Error: stray '@' in program (line: 2)
3	@\$d Error: stray '\$' in program (line: 3) d - id	Error: stray '@' in program (line: 3)

## 2.3 Número malformatado

Esse tipo de erro só acontece quando o analisador se depara com algo que começa com um número e é seguido por uma letra, nesse caso é relatado um erro de número malformatado. A Tabela 2 mostra alguns exemplos de código fonte (Entrada) e de saída do analisador (Saída) para exemplificar como ele se comporta nesse contexto.

Tabela 2: Caso de teste *input1.txt* para reconhecimento de números e números mal formados.

Entrada	Saída
123	123 - number_int
123.123	123.123 - number_real
123asd	Error: malformed number '123asd' in program (line: 3)
123.123asd	Error: malformed number '123.123asd' in program (line: 4)
123.123@sd	123.123 - number_real Error: stray '@' in program (line: 5) sd - id
123.123.123	123.123 - number_real . - . 123 - number_int

Um possível erro que pensamos em relatar é o de número com um tamanho muito grande de dígitos, entretanto vimos que o GCC aponta isso como um warning e não um erro. Por isso optamos por não tratar isso na análise léxica.

## 2.4 Identificador malformatado

O analisador não relata erros de identificador malformatado, assim como no GCC, um identificador é lido até o ponto onde se encontra outro *token* ou um símbolo que não pertence ao alfabeto da linguagem. A Tabela 3 mostra alguns exemplos de código fonte (Entrada) e de saída do analisador (Saída) para exemplificar como ele se comporta nesse contexto.

Tabela 3: Caso de teste *input2.txt* para reconhecimento de identificadores.

Entrada	Saída
asd	asd - id
asd.qwe	asd - id · - · qwe - id
asd=qwe	asd - id = - = qwe - id
asd@qwe	asd - id Error: stray '@' in program (line: 4) qwe - id