

# GCC130 Compiladores

## Relatório da Entrega 1: Analisador Léxico

Bernado Diniz  
Luan Shimosaka  
Luiz Phillip

Setembro de 2025

# Sumário

<b>1</b>	<b>Decisões de Projeto</b>	<b>3</b>
<b>2</b>	<b>Dificuldades Encontradas</b>	<b>3</b>
<b>3</b>	<b>Diagramas de Transição (DFAs)</b>	<b>4</b>
3.1	DFA para Identificadores (ID) . . . . .	4
3.2	DFA para Números Inteiros (NUMBER) . . . . .	4
<b>4</b>	<b>Testes e Validação</b>	<b>5</b>
4.1	Arquivo de Teste (teste.bll) . . . . .	5
4.2	Saída do Analisador . . . . .	5

# 1 Decisões de Projeto

Durante o desenvolvimento do analisador léxico, foram tomadas as seguintes decisões de projeto para atender aos requisitos de forma clara e eficiente:

- **Tabela de Símbolos:** Optou-se por uma implementação com um vetor estático e busca linear. Essa abordagem é simples e performática para o escopo do trabalho, que não espera um número massivo de identificadores distintos. A função de inserção garante que os identificadores não sejam duplicados, armazenando apenas a primeira ocorrência.
- **Gerenciamento de Posição:** Para o rastreamento da coluna, uma variável global (`'column_number'`) foi atualizada manualmente a cada token reconhecido. A variável `'yylineno'`, fornecida nativamente pelo Flex, foi utilizada para o controle da linha. Essa combinação permite reportar a posição exata de tokens e erros.
- **Expressões Regulares:** As expressões regulares foram definidas de forma a priorizar as palavras-chave sobre a regra genérica de identificadores (`'id'`). Isso foi feito simplesmente listando as regras das palavras-chave antes da regra do `'id'`, aproveitando o comportamento padrão do Flex de casar a regra mais longa e, em caso de empate, a que aparece primeiro.
- **Tratamento de Erros:** Foi definida uma regra genérica (`'.'`) ao final do arquivo para capturar quaisquer caracteres que não se encaixem nos tokens definidos. Essa regra imprime uma mensagem de erro clara no `'stderr'`, informando a posição exata da falha, sem interromper a análise do restante do arquivo.

# 2 Dificuldades Encontradas

O desenvolvimento apresentou alguns desafios, que foram superados com as seguintes soluções:

- **Regex para Comentários de Bloco:** A elaboração da expressão regular para comentários de múltiplas linhas (`'/* ... */'`) foi a tarefa mais complexa. Foi necessário garantir que ela tratasse corretamente casos de borda, como múltiplos asteriscos (`'/**/'`) ou conteúdo que se assemelhasse ao fechamento do bloco, sem causar falhas no reconhecimento.
- **Integração de Código C:** A integração das funções em C para a tabela de símbolos dentro da seção de definições do Flex exigiu atenção à sintaxe e ao gerenciamento de memória. O uso da função `'strdup()'` foi essencial para alocar memória dinamicamente para os lexemas, evitando que o ponteiro `'yytext'` fosse sobrescrito em leituras subsequentes.

### 3 Diagramas de Transição (DFAs)

A seguir, são apresentados os diagramas de autômatos finitos para duas das principais classes de tokens da linguagem.

#### 3.1 DFA para Identificadores (ID)

Este autômato reconhece identificadores que devem começar com uma letra e podem ser seguidos por letras ou dígitos.

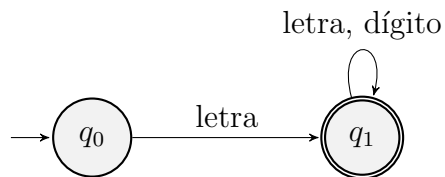


Figura 1: Diagrama de Autômato Finito para o token ID.

#### 3.2 DFA para Números Inteiros (NUMBER)

Este autômato reconhece sequências de um ou mais dígitos, correspondendo a números inteiros.

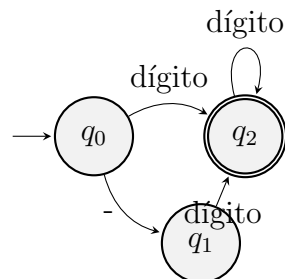


Figura 2: Autômato para inteiros (positivos ou negativos).

## 4 Testes e Validação

Para validar o analisador, foi utilizado um arquivo "teste.bl".

### 4.1 Arquivo de Teste (teste.bl)

```
/*
 * Arquivo de teste para o Analisador Lexico
 * da linguagem "mini C".
 */
int main() {
    int a;
    int b = -100;
    bool flag = true;

    // Teste de operadores e expressoes
    a = b * 2;
    if (a < 0 && flag != false) {
        print(a);
    }

    /* Teste de laço e erro lexico */
    int count = 5;
    while (count > 0) {
        count = count - 1;
    }

    // O caractere '$' deve gerar um erro.
    int erro = 5$;
}
```

### 4.2 Saída do Analisador

```
youserz@fedora:~/Documentos/programas/ClimateDataStore$ ./main teste.bll
(base) youserz@fedora:~/Documentos/programas/ClimateDataStore$ ./main teste.bll

<INT, int, 5, 1> <ID, main, 5, 5> <LPARENTESE, (, 5, 10> <RPARENTESE, ), 5, 11> <LCHAVES, {, 5, 13>
<INT, int, 6, 1> <ID, a, 6, 5> <SEMICOLON, ;, 6, 7>
<INT, int, 7, 1> <ID, b, 7, 5> <ASSIGN, =, 7, 7> <NUMBER, -100, 7, 9> <SEMICOLON, ;, 7, 13>
<BOOL, bool, 8, 1> <ID, flag, 8, 6> <ASSIGN, =, 8, 11> <TRUE, true, 8, 13> <SEMICOLON, ;, 8, 18>

<ID, a, 10, 1> <ASSIGN, =, 10, 3> <ID, b, 10, 5> <TIMES, *, 10, 7> <NUMBER, 2, 10, 9> <SEMICOLON, ;, 10, 10>
<IF, if, 11, 1> <LPARENTESE, (, 11, 4> <ID, a, 11, 6> <LT, <, 11, 8> <NUMBER, 0, 11, 10> <AND, &&, 11, 12> <ID, flag, 11, 15> <NE, !=, 11, 20> <FALSE,
false, 11, 23> <RPARENTESE, ), 11, 29> <LCHAVES, {, 11, 31>
<PRINT, print, 12, 1> <LPARENTESE, (, 12, 7> <ID, a, 12, 9> <RPARENTESE, ), 12, 11> <SEMICOLON, ;, 12, 13>
<RCHAVES, }, 13, 1>

<INT, int, 15, 1> <ID, count, 15, 5> <ASSIGN, =, 15, 11> <NUMBER, 5, 15, 13> <SEMICOLON, ;, 15, 14>
<WHILE, while, 16, 1> <LPARENTESE, (, 16, 7> <ID, count, 16, 9> <GT, >, 16, 15> <NUMBER, 0, 16, 17> <RPARENTESE, ), 16, 18> <LCHAVES, {, 16, 20>
<ID, count, 17, 1> <ASSIGN, =, 17, 7> <ID, count, 17, 9> <MINUS, -, 17, 15> <NUMBER, 1, 17, 17> <SEMICOLON, ;, 17, 18>
<RCHAVES, }, 18, 1>

Erro lexico na linha 20, coluna 14. Caractere desconhecido: "$"
<INT, int, 20, 1> <ID, erro, 20, 5> <ASSIGN, =, 20, 10> <NUMBER, 5, 20, 12> <SEMICOLON, ;, 20, 16>
<RCHAVES, }, 21, 1>

--- Tabela de Símbolos ---
Lexema      Posição (Linha, Coluna)
-----
main        (5, 5)
a           (6, 5)
b           (7, 5)
flag        (8, 6)
count       (15, 5)
erro        (20, 5)

(base) youserz@fedora:~/Documentos/programas/ClimateDataStore$
```

Figura 3: Saída gerada pelo analisador léxico.