GCC130 Compiladores
Relatório da Entrega 1: Analisador Léxico

Bernado Diniz
Luan Shimosaka
Luiz Phillip

Setembro de 2025

Sumário

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.

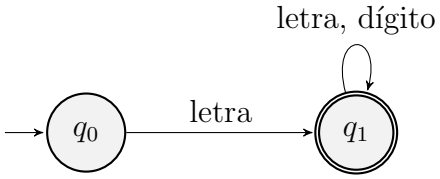

Figura 1: Diagrama de Autômato Finito para o token ID.

3.2 DFA para Números Inteiros (NUMBER)

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

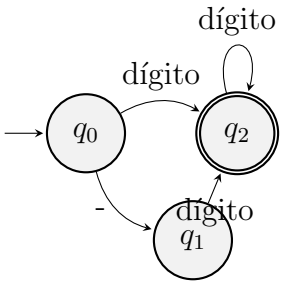

Figura 2: Autômato para inteiros (positivos ou negativos).

4 Testes e Validação

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

4.1 Arquivo de Teste (teste.bll)

```
/*
 * 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 laco 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

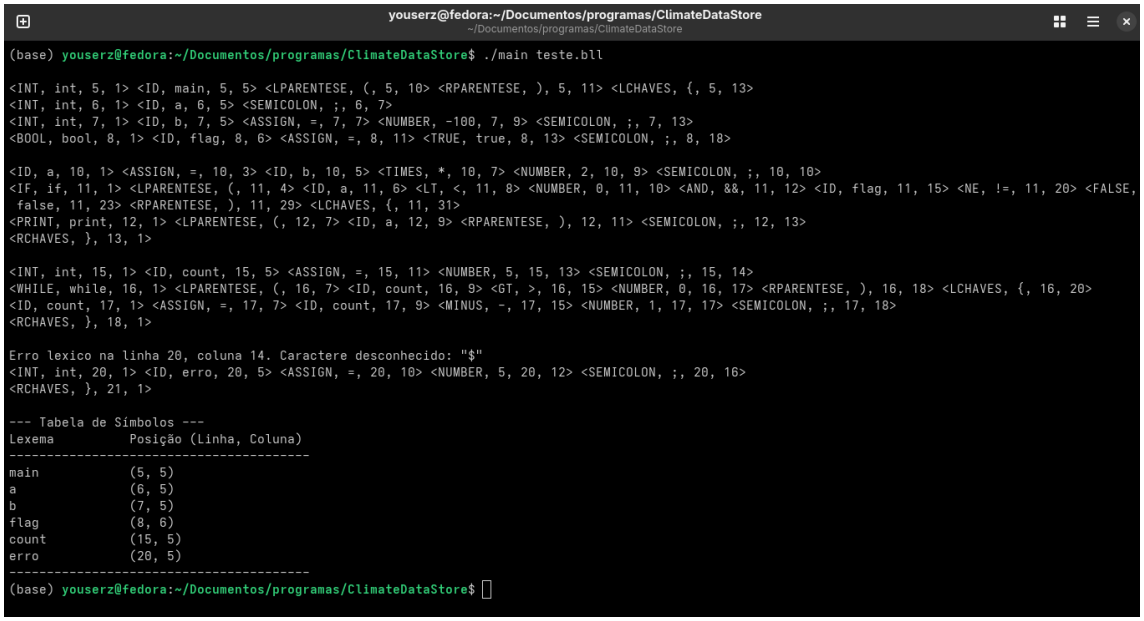

Figura 3: Saída gerada pelo analisador léxico.