INSTITUTO SUPERIOR POLITÉCNICO DE TETE

DIVISÃO DE ENGENHARIA

COMPILADORES

Turma: `A`

Tema: **Análise sintática**

De:

Antonio da Ester

Alaín neves

Iverson Machava

Nilton maranda

Tete, outubro de 2025

De:

Antonio da Ester

Alaín neves

Iverson Machava

Nilton maranda

Tema: **Análise sintática**

Primeiro trabalho de pesquisa para a cadeira de Compiladores, terceiro ano, primeiro semestre, sobre a arquitetura dos Sistemas distribuídos, dado pelo docente:

Docente: Lourenço Roberto

Tete, outubro de 2025

Índice

CAPITUO I

1. Introdução

A comunicação entre o ser humano e o computador é feita por meio de linguagens formais — conjuntos de regras bem definidas que permitem escrever programas que uma máquina pode entender. No entanto, para que o computador consiga interpretar corretamente essas instruções, é necessário traduzi-las em uma estrutura compreensível. Esse é o papel da análise sintáctica.

A análise sintáctica é uma das etapas mais importantes na construção de um compilador ou interpretador. Ela verifica se a sequência de comandos escrita pelo programador segue as regras da linguagem, permitindo identificar erros e compreender a estrutura do código.

automatizar essa tarefa está o Bison, um gerador de analisadores sintácticos do projeto GNU. Em conjunto com o Flex, um analisador léxico, o Bison forma uma poderosa dupla para o desenvolvimento de linguagens, interpretadores e compiladores.

O presente trabalho tem como objetivo explorar o funcionamento da ferramenta Bison e a sua integração com o Flex, demonstrando como ambas atuam na análise e interpretação de linguagens de programação.

2. Objectivos

2.1. Objectivo Geral

Estudar e compreender o funcionamento da ferramenta Bison e sua integração com o Flex, analisando o papel de cada uma no processo de análise de linguagens formais e sua aplicação prática na construção de sistemas de interpretação e compilação deprogramação.

2.2. Objectivos Específicos

- Descrever o conceito e o funcionamento da análise sintáctica.
- Explicar a estrutura e os principais componentes da ferramenta Bison.
- Demonstrar como ocorre a integração entre o Bison e o Flex no processo de compilação.
- Apresentar exemplos práticos de utilização dessas ferramentas.
- Identificar as vantagens, limitações e aplicações práticas da sua utilização conjunta.
- Desenvolver uma visão crítica sobre o papel dessas ferramentas no ensino e desenvolvimento de compiladores modernos.

3. Metodologia

Este trabalho foi desenvolvido com base em pesquisa bibliográfica e prática experimental.

1. Pesquisa Bibliográfica:
   Foram consultadas fontes académicas, manuais técnicos e documentação oficial das ferramentas GNU Bison e Flex. O estudo abrangeu desde os fundamentos teóricos da análise sintáctica até exemplos de implementações práticas em C.
2. Análise Experimental:
   Foi criado um exemplo prático de implementação de uma mini calculadora utilizando o Bison e o Flex, com o objetivo de compreender de forma aplicada o processo de integração entre o analisador léxico e o sintáctico.
3. Síntese dos Resultados:
   A partir da observação dos testes realizados e das fontes estudadas, foi elaborada uma análise crítica sobre a importância, as vantagens e as limitações dessas ferramentas.

CAPITULO II

1. Revisão bibliográfica

1.1.A Ferramenta Bison e o Processo de Análise Sintáctica

Bison é um gerador de analisadores poderoso e amplamente utilizado que faz parte da GNU Compiler Collection (GCC). Ele é baseado no gerador de analisadores YACC (Yet Another Compiler Compiler), mas inclui uma série de melhorias e aprimoramentos.

O Bison utiliza uma gramática livre de contexto (CFG) para descrever a sintaxe da linguagem analisada e pode gerar um analisador sintático em C ou C++ que pode ser usado em diversas aplicações, incluindo compiladores, interpretadores e ferramentas de análise de código-fonte. O Bison é conhecido por seu alto desempenho e flexibilidade, bem como por sua extensa documentação e comunidade de usuários ativa.

1.2.**Conceitos Fundamentais**

Antes de entender o funcionamento do Bison, é importante conhecer alguns conceitos básicos:

- Gramática Livre de Contexto (GLC): conjunto de regras que descrevem como uma linguagem é estruturada.
- Tokens: são as menores unidades de significado reconhecidas pelo computador, como números, palavras-chave e operadores.
- Parser: é o componente responsável por analisar a sequência de tokens e verificar se ela segue as regras da gramática.
- Ações semânticas: pequenos blocos de código que determinam o que deve ser feito quando uma regra gramatical é reconhecida (por exemplo, somar dois números).
- Shift e Reduce: operações internas do parser que permitem “empilhar” e “reduzir” símbolos até formar estruturas válidas da linguagem.

O Bison utiliza um método chamado LALR(1), que é eficiente e amplamente adotado em compiladores modernos.

1.3.**Como o Bison Funciona?**

O funcionamento do Bison é baseado num arquivo de entrada com extensão .y, onde o programador descreve a gramática da linguagem que deseja interpretar. Esse arquivo é dividido em três partes principais:

1. Declarações: onde são definidos os tokens e suas características (por exemplo, operadores matemáticos).
2. Regras de Produção: que descrevem como as expressões válidas da linguagem são formadas.
3. Ações Semânticas: onde se define o que fazer quando uma regra é reconhecida — por exemplo, calcular o valor de uma expressão.

Após escrever esse arquivo, o programador executa o comando bison nome_do_arquivo.y. O Bison então gera automaticamente um código em C (ou C++) com uma função chamada yyparse(), que faz toda a análise sintáctica. Normalmente, esse parser é usado junto com uma ferramenta chamada Flex, que faz a análise léxica (identifica os tokens).

4.1. Exemplo Prático

Exemplo simples de gramática para expressões aritméticas:

```
%{
#include <stdio.h>
#include <stdlib.h>
int yylex(void);
int yyerror(char *s);
%}

%token NUMBER
%left '+' '-'
%left '*' '/'
```

```
%%
expr:
    expr '+' expr   { printf("%d\n", $1 + $3); }
  | expr '-' expr   { printf("%d\n", $1 - $3); }
  | expr '*' expr   { printf("%d\n", $1 * $3); }
  | expr '/' expr   { printf("%d\n", $1 / $3); }
  | '(' expr ')'    { $$ = $2; }
  | NUMBER          { $$ = $1; }
  ;
%%
int main() {
    printf("Digite uma expressão: ");
    yyparse();
    return 0;
}

int yyerror(char *s) {
    printf("Erro: %s\n", s);
    return 0;
}

int yylex(void) {
    int c;
    while ((c = getchar()) == ' ' || c == '\t');
    if (c == EOF)
        return 0;
    if (isdigit(c)) {
        ungetc(c, stdin);
        scanf("%d", &yylval);
        return NUMBER;
    }
    return c;
}
```

Com este programa, o Bison permite processar e avaliar expressões como: 3 + 5 * 2 → o resultado seria 13, pois o parser respeita a precedência dos operadores.

1.4.Vantagens e desvantagens do uso do Bison

1.4.1. Vantagens

Automatização Gera código de parser automaticamente a partir da gramática, poupando tempo e reduzindo erros manuais.

Precisão Bison detecta ambiguidades e conflitos, alertando o desenvolvedor.

Performance Parsers gerados por Bison (LALR(1)) são eficientes para muitas linguagens práticas.

Flexibilidade semântica Permite definir ações e construir ASTs ou calcular valores diretamente.

1.4.2. **Desvantagens**

- **Complexidade para gramáticas grandes** gramáticas complexas podem gerar muitos conflitos; debugging pode ser desafiador.
- **Mensagens de erro menos amigáveis** às vezes o erro sintáctico reportado está distante do problema real.
- **Dependência de ferramentas externas (lexer)** normalmente se usa flex ou algo similar; precisa de mais partes para compilar tudo.
- **Restrição de expressividade** algumas linguagens com gramáticas contextuais ou sensíveis ao contexto exigem truques ou extensões.
- **Curva de aprendizagem** é necessário entender teoria de parsing, gramáticas, tokens, conflitos, precedência, etc.

1.5.**Aplicações Práticas**

O Bison é utilizado em diversos contextos, entre eles:

- Criação de compiladores e interpretadores de linguagens de programação.
- Desenvolvimento de linguagens específicas de domínio (DSLs), usadas em áreas técnicas.
- Análise de dados estruturados, como JSON e XML.
- Ferramentas de análise de código e verificação de erros.

2. A Combinação entre o Bison e o Flex

Flex e Bison são uma combinação popular de ferramentas para construir compiladores e outras ferramentas de linguagem. Flex (Fast Lexical Analyzer Generator) é uma ferramenta para gerar scanners, que são usados para decompor o texto de entrada em tokens, enquanto Bison é uma ferramenta para gerar analisadores sintáticos, que são usados para analisar a estrutura do texto de entrada com base em uma gramática livre de contexto.

2.1. Como o Flex e o Bison se Integram?

1. O Flex processa um arquivo .l contendo regras de padrões (usando expressões regulares). Ele gera um código em C com uma função yylex() que identifica tokens.
2. O Bison processa um arquivo .y com as regras da gramática e gera o código yyparse() que usa a função yylex() para obter os tokens.
3. Juntos, eles formam um sistema completo de análise léxica e sintáctica.

2.2.Exemplo Prático

3. Lexer.l:
4. %{
5. #include "calc.tab.h"
6. %}
7.
8. %%
9. [0-9]+ { yylval = atoi(yytext); return NUMBER; }

```
10. [ \t\n]     ;  // Ignora espaços
11. [+\-*/()]   { return yytext[0]; }
12. %%
13. Analisador:
14. %{
15. #include <stdio.h>
16. #include <stdlib.h>
17. int yylex(void);
18. void yyerror(const char *s);
19. %}
20.
21. %token NUMBER
22. %left '+' '-'
23. %left '*' '/'
24.
25. %%
26. expr:
27.     expr '+' expr   { $$ = $1 + $3; }
28.   | expr '-' expr   { $$ = $1 - $3; }
29.   | expr '*' expr   { $$ = $1 * $3; }
30.   | expr '/' expr   { $$ = $1 / $3; }
31.   | '(' expr ')'    { $$ = $2; }
32.   | NUMBER          { $$ = $1; }
33.   ;
34. %%
35. int main() { return yyparse(); }
36. void yyerror(const char *s) { fprintf(stderr, "Erro: %s\n", s); }
37.
```

2.3. Vantagens da Combinação

- **Automatização total** o programador não precisa escrever manualmente o código de análise; tudo é gerado a partir das regras.
- **Separação de responsabilidades** o Flex cuida do texto bruto; o Bison cuida da lógica e estrutura.
- **Performance** os analisadores gerados são rápidos e eficientes.
- **Escalabilidade** é fácil expandir a linguagem, adicionar novas regras ou operadores.
- **Robustez** detecta erros sintácticos e permite recuperação parcial da entrada.

CAPITULO III

1. Conclusão

A integração entre o Bison e o Flex representa uma das abordagens mais eficazes e didáticas para a construção de analisadores de linguagens formais.

Enquanto o Flex realiza a leitura e classificação dos tokens, o Bison interpreta a estrutura e o significado das instruções, formando uma parceria que espelha o funcionamento real dos compiladores modernos.

O domínio dessas ferramentas não apenas amplia o conhecimento sobre o funcionamento das linguagens de programação, mas também fortalece as competências técnicas para o desenvolvimento de software avançado.

Assim, o estudo dessas ferramentas é essencial para estudantes e profissionais da área de engenharia de software e computação.

2. Referências Bibliográficas

- GNU Bison Manual. https://www.gnu.org/software/bison
- Flex Manual. https://westes.github.io/flex/manual/
- Aho, A. V., Lam, M., Sethi, R., & Ullman, J. D. *Compiladores: Princípios, Técnicas e Ferramentas.* Pearson, 2008.
- Thiago H. “Introdução ao Bison e Flex.” https://thiagoh.github.io/bison