INSTITUTO SUPERIOR POLITÉCNICO DE TETE

DIVISÃO DE ENGENHARIA

ENGENHARIA INFORMÁTICA

Turma: B

Compiladores

Tema: Análise Sintáctica em Bison

Estudantes:

Jéssica Rosário

Matias Alberto Matavel

Nélcio Juanete

Taily Come

Tete, Outubro de 2025

DIVISÃO DE ENGENHARIA

Jéssica Rosário

Matias Alberto Matavel

Nélcio Juanete

Taily Come

Análise Sintáctica em Bison

Trabalho de carácter avaliativo na cadeira de Compiladores

Docente: Msc. Lourenço Roberto

Tete, Outubro de 2025

Índice

1. Introdução

A análise sintáctica é uma das etapas fundamentais no processo de compilação, responsável por verificar se a sequência de tokens produzida pelo analisador léxico (scanner) está de acordo com as regras gramaticais da linguagem. O *GNU Bison* é uma das ferramentas mais utilizadas para gerar analisadores sintácticos automáticos, capazes de interpretar linguagens formais, linguagens de programação, expressões matemáticas e até comandos estruturados. Este trabalho tem como foco o estudo e aplicação da análise sintáctica utilizando o Bison, explorando duas abordagens distintas: com o uso da ferramenta *Flex* (para análise léxica) e sem o uso dela (com o scanner manual implementado em C/C++). Assim, busca-se compreender o funcionamento interno da comunicação entre o analisador léxico e o sintáctico, bem como suas vantagens, limitações e aplicabilidades em projectos reais.

2. Objectivos

2.1. Objectivo Geral

Investigar o funcionamento do analisador sintáctico gerado pelo Bison, compreendendo sua integração com o analisador léxico (Flex) e o seu uso independente, de modo a demonstrar suas principais características, vantagens e aplicações práticas.

2.2. Objectivos Específicos

- Explicar o processo de análise sintáctica e o papel do Bison na geração de parsers.
- Implementar exemplos práticos de analisadores sintácticos com e sem Flex.
- Comparar as vantagens, desvantagens e contextos de uso de cada abordagem.
- Demonstrar a importância da análise sintáctica na construção de linguagens de programação e interpretadores.
- Identificar possíveis aplicações do Bison em projectos de compiladores e processadores de comandos.

3. Metodologia

A metodologia adoptada neste trabalho é de carácter **exploratório e experimental**. Primeiramente, foi realizada uma pesquisa bibliográfica e documental sobre o funcionamento do Bison e sua integração com o Flex, por meio de manuais oficiais, artigos e exemplos práticos disponíveis na literatura e comunidades de desenvolvedores.
Em seguida, foram desenvolvidos e testados **dois protótipos de analisadores sintáticos**:

1. Um utilizando **Bison e Flex**, para demonstrar a comunicação entre o analisador léxico e o sintático;
2. Outro utilizando **Bison sem Flex**, onde o analisador léxico foi implementado manualmente em C++, reforçando o entendimento do fluxo interno de tokens.

Os exemplos foram implementados, compilados e executados em ambiente **Dev-C++/GCC**, com base em gramáticas simples, como a de uma calculadora aritmética. Por fim, os resultados foram comparados e analisados qualitativamente quanto à facilidade de desenvolvimento, desempenho e flexibilidade.

4. Fundamentação Teórica

4.1. Análise Sintáctica e o Papel do Bison

O Bison é um gerador de analisadores sintácticos baseado em gramáticas livres de contexto. A ferramenta traduz uma gramática formal, descrita em um arquivo .y, em código C ou C++ capaz de processar entradas de texto de acordo com essas regras.
Sua principal função é construir um **parser**, que recebe tokens do analisador léxico e verifica se eles formam uma estrutura válida conforme a gramática definida. O Bison é amplamente utilizado em compiladores, interpretadores, processadores de linguagem e sistemas que dependem de interpretação de comandos.

4.2. Uso do Bison com Flex

a) Integração entre Flex e Bison

O *Flex* (Fast Lexical Analyzer) é uma ferramenta usada para gerar analisadores léxicos — programas que dividem uma sequência de caracteres em *tokens*, que são então processados pelo Bison.
O fluxo de trabalho típico é:

1. O Flex lê um arquivo .l e gera o código-fonte do analisador léxico (lex.yy.c);
2. O Bison lê um arquivo .y e gera o analisador sintáctico (parser.tab.c e parser.tab.h);
3. Ambos os códigos são compilados juntos e o resultado é um programa capaz de reconhecer e processar uma linguagem.

b) Vantagens do uso conjunto (Flex + Bison):

- **Automação completa:** o Flex e o Bison automatizam as etapas de análise léxica e sintáctica.
- **Manutenção simplificada:** alterações na gramática ou nas regras léxicas podem ser feitas directamente nos arquivos .y e .l.
- **Desempenho optimizado:** ambos geram código em C altamente eficiente.

- **Amplamente adoptado:** é o padrão em projectos académicos e profissionais de compiladores.

c) Aplicações e Casos de Uso

- Compiladores de linguagens de programação (ex.: C, Pascal, SQL).
- Interpretadores e shells de comandos.
- Ferramentas de análise e transformação de código (linters, formatação, pré-processadores).
- Sistemas que processam linguagens de domínio específico (DSLs).

Exemplo de Implementação do Bison com Flex:

```
/*  ANALISADOR LÉXICO (Flex)  */

%{

#include <stdio.h>

#include "calc.tab.h"

%}

%%

[0-9]+          { yylval.num = atoi(yytext); return NUMERO; }

"+"             { return SOMA; }

"-"             { return SUBTRACAO; }

"*"             { return MULTIPLICACAO; }

"/"             { return DIVISAO; }

\n              { return NOVA_LINHA; }

[ \t]           { /* ignorar espaços */ }

.               { printf("Erro: caractere inválido\n"); } %%
```

```
int yywrap() { return 1; }

/* ===== ANALISADOR SINTÁTICO (Bison) ===== */

%{

#include <stdio.h>

extern int yylex();

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

%}

%token NUMERO SOMA SUBTRACAO MULTIPLICACAO DIVISAO NOVA_LINHA

%left SOMA SUBTRACAO

%left MULTIPLICACAO DIVISAO

%%

programa:

    programa declaracao

    | declaracao

    ;

declaracao:

    expressao NOVA_LINHA { printf("Resultado: %d\n", $1); }

    ;

expressao:

    expressao SOMA termo { $$ = $1 + $3; }

    | expressao SUBTRACAO termo { $$ = $1 - $3; }

    | termo { $$ = $1; }   ;
```

```
termo:
termo MULTIPLICACAO fator { $$ = $1 * $3; }
    | termo DIVISAO fator { $$ = $1 / $3; }
    | fator { $$ = $1; }
    ;
fator:
    NUMERO { $$ = $1; }
    ;
%%
/* ===== PROGRAMA PRINCIPAL ===== */
int main() {
    printf("=== CALCULADORA COM BISON + FLEX ===\n");
    printf("Digite expressões como: 2 + 3 * 4\n");
    printf("Pressione Ctrl+C para sair\n\n");
        yyparse();
    return 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
```

4.3. Uso do Bison sem Flex

a) Abordagem sem Flex

É possível usar o Bison sem o Flex implementando manualmente a função yylex(), responsável por identificar tokens. Neste caso, o programador lê a entrada caractere por caractere (ou palavra por palavra) e retorna tokens conforme as regras desejadas.
O parser gerado pelo Bison continua a funcionar da mesma forma, recebendo tokens e aplicando as regras da gramática.

b) Vantagens do uso isolado (Bison sem Flex):

- **Maior controle:** o programador define manualmente como a análise léxica deve ocorrer.
- **Simplicidade:** ideal para projectos pequenos, protótipos e aprendizagem.
- **Sem dependências externas:** não é necessário instalar ou configurar o Flex.
- **Flexibilidade de linguagem:** o código pode ser ajustado directamente em C ou C++, facilitando integração com outros sistemas.

c) Aplicações e Casos de Uso

- Pequenas calculadoras e interpretadores simples.
- Verificação sintáctica de expressões matemáticas, lógicas ou condicionais.
- Processamento de comandos em sistemas embarcados ou educativos.
- Protótipos didácticos para ensino de compiladores.

d) Limitações

- Maior esforço para implementar e manter a função yylex().
- Menos eficiência para linguagens complexas.
- Risco de inconsistência se o scanner manual não corresponder exactamente às expectativas do parser.

Exemplo de Implementação do Bison sem Flex:

Criação dum analisador sintáctico que entenda expressões aritméticas com +, -, *, / e parênteses, respeitando precedência dos operadores.

Considerando os seguintes ficheiros:

calc.y:

```
 %{
 #include <stdio.h>
#include <stdlib.h>
// Declara a função de erro
void yyerror(const char *s);
int yylex(void); %}
// Definição dos tokens
% token NUM
// Definindo precedência dos operadores
%left '+' '-'
%left '*' '/'
%left UMINUS
%%
// Regras da gramática
input:
    /* vazio */
  | input line ;
line:  '\n'
  | expr '\n'    { printf("Resultado = %d\n", $1); } ;
expr:
```

```
   NUM                 { $$ = $1; }
  | expr '+' expr      { $$ = $1 + $3; }
  | expr '-' expr      { $$ = $1 - $3; }
  | expr '*' expr      { $$ = $1 * $3; }
  | expr '/' expr      {  if ($3 == 0) { yyerror("Divisão por zero!"); $$ = 0; }
else {  $$ = $1 / $3;} }
  | '-' expr %prec UMINUS { $$ = -$2; }
  | '(' expr ')'       { $$ = $2; }
  ; %%

// Função de erro
void yyerror(const char *s) { fprintf(stderr, "Erro de sintaxe: %s\n", s);}
```

main.cpp:

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
// Inclui os tokens e declarações do parser
#include "C:\\Users\\User\\Documents\\projecto_bison\\calc.tab.h"

// Declarações que vêm do parser (calc.tab.c)
int yyparse(); void yyerror(const char *s);
// Implementação simples de yylex (sem Flex)
```

```
int yylex() {    int c;


    // Ignora espaços e tabulações
    while ((c = getchar()) == ' ' || c == '\t');
    // Fim do arquivo (CTRL+Z no Windows)
    if (c == EOF)    return 0;
    // Se for número
    if (isdigit(c)) {
 ungetc(c, stdin); // devolve o caractere
        int num;
       scanf("%d", &num);
        yylval = num; // passa o valor para o parser
        return NUM;   // <<< Agora o NUM está declarado em calc.tab.h
    }
    // Caso contrário, retorna o próprio caractere (operador ou parêntese)
    return c;
}
// Função principal
int main() {
    printf("=== Calculadora em Bison (C++) ===\n");
    printf("Digite uma expressao e pressione Enter.\n");
    yyparse(); // chama o parser
  return 0; }
```

Fluxo de Implementação:

1. A gramática é escrita num ficheiro do tipo y, neste caso é no calc.y.
2. Executa bison -d calc.y → gera calc.tab.c (implementação do parser) e calc.tab.h (tokens e declarações).
3. O yylex() (o *scanner*, aqui no main.cpp) que lê caracteres/strings e devolve tokens para o parser, é implementado.
4. yyparse() (do calc.tab.c) chama repetidamente yylex() para obter tokens e aplica as regras da gramática. Quando uma regra tem uma ação ({ ... }), o código é executado (por exemplo, calcular soma).
5. Saída: o seu programa mostra o resultado ou mensagem de erro via yyerror().

5. Conclusão

O estudo sobre análise sintáctica com o Bison demonstra a importância desta ferramenta no desenvolvimento de compiladores e processadores de linguagem. O uso combinado de Bison e Flex fornece uma solução automatizada, robusta e eficiente, ideal para linguagens mais complexas e projectos de médio e grande porte.

Por outro lado, o uso do Bison sem o Flex é uma abordagem valiosa para fins didácticos, experimentais ou quando se deseja controle total sobre a análise léxica. Essa abordagem ajuda o estudante a compreender o funcionamento interno do processo de compilação e a comunicação entre o analisador léxico e o sintáctico.

Ambas as abordagens evidenciam a versatilidade e a importância do Bison como ferramenta educacional e profissional, essencial no estudo da análise sintáctica e na construção de linguagens formais.

6. Referências

- GNU Project — *Bison Manual*. Disponível em: https://www.gnu.org/software/bison/manual/
- GNU Project — *Flex Manual*. Disponível em: https://westes.github.io/flex/manual/
- AHO, A. V.; LAM, M. S.; SETHI, R.; ULLMAN, J. D. **Compiladores: Princípios, Técnicas e Ferramentas**. 2ª ed. Pearson, 2008.
- COOPER, K.; TORCZON, L. **Engineering a Compiler**. 2nd ed. Morgan Kaufmann, 2011.
- GESHI, A. **An Introduction to Bison and Flex**. Linux Journal, 2010.
- WATTERS, C. **Using Flex and Bison for Parsing and Lexical Analysis**. TutorialsPoint, 2023.
- DEV Community. *How to Build a Calculator Using Bison Without Flex*. Disponível em: https://dev.to.
- Stack Overflow — Discussões sobre integração Flex/Bison e implementação manual de yylex().