

# GCC130 Compiladores

## Relatório da Entrega 2: Analisador Sintático

Bernado Diniz  
Luan Shimosaka  
Luiz Phillip

Outubro de 2025

# Sumário

<b>1</b>	<b>Gramática da Linguagem (BNF)</b>	<b>3</b>
<b>2</b>	<b>Decisões de Projeto</b>	<b>4</b>
<b>3</b>	<b>Dificuldades Encontradas</b>	<b>5</b>
<b>4</b>	<b>Conjuntos FIRST e FOLLOW</b>	<b>5</b>
<b>5</b>	<b>Análise de Conflitos do Autômato</b>	<b>5</b>
5.1	Conflito "Dangling Else	5
5.2	Conflito do Menos Unário (-) . . . . .	6
<b>6</b>	<b>Testes e Validação</b>	<b>8</b>
6.1	Arquivo de Teste Sintático (teste_sintatico.bll) . . . . .	8
6.2	Saída do Analisador . . . . .	8

# 1 Gramática da Linguagem (BNF)

$\langle \text{program} \rangle ::= \langle \text{declaration\_list} \rangle$

$\langle \text{declaration\_list} \rangle ::= \langle \text{declaration\_list} \rangle \langle \text{declaration} \rangle$   
|  $\varepsilon$

$\langle \text{declaration} \rangle ::= \langle \text{variable\_declaration} \rangle$   
|  $\langle \text{function\_declaration} \rangle$

$\langle \text{variable\_declaration} \rangle ::= \langle \text{type} \rangle \text{ T\_ID T\_SEMICOLON}$   
|  $\langle \text{type} \rangle \text{ T\_ID T\_ASSIGN } \langle \text{expression} \rangle \text{ T\_SEMICOLON}$

$\langle \text{function\_declaration} \rangle ::= \langle \text{type} \rangle \text{ T\_ID T\_LPARENTESE T\_RPARENTESE}$   
 $\langle \text{block\_statement} \rangle$

$\langle \text{block\_statement\_list} \rangle ::= \langle \text{block\_statement\_list} \rangle \langle \text{statement} \rangle$   
|  $\varepsilon$

$\langle \text{statement} \rangle ::= \langle \text{variable\_declaration} \rangle$   
|  $\langle \text{assignment\_statement} \rangle$   
|  $\langle \text{if\_statement} \rangle$   
|  $\langle \text{while\_statement} \rangle$   
|  $\langle \text{io\_statement} \rangle$   
|  $\langle \text{block\_statement} \rangle$

$\langle \text{type} \rangle ::= \text{ T\_INT}$   
|  $\text{ T\_BOOL}$

$\langle \text{assignment\_statement} \rangle ::= \text{ T\_ID T\_ASSIGN } \langle \text{expression} \rangle \text{ T\_SEMICOLON}$

$\langle \text{if\_statement} \rangle ::= \text{ T\_IF T\_LPARENTESE } \langle \text{expression} \rangle \text{ T\_RPARENTESE}$   
 $\langle \text{statement} \rangle$   
|  $\text{ T\_IF T\_LPARENTESE } \langle \text{expression} \rangle \text{ T\_RPARENTESE } \langle \text{statement} \rangle$   
 $\text{ T\_ELSE } \langle \text{statement} \rangle$

$\langle \text{while\_statement} \rangle ::= \text{ T\_WHILE T\_LPARENTESE } \langle \text{expression} \rangle \text{ T\_RPARENTESE}$   
 $\langle \text{statement} \rangle$

$\langle \text{block\_statement} \rangle ::= \text{ T\_LCHAVES } \langle \text{block\_statement\_list} \rangle \text{ T\_RCHAVES}$

$\langle \text{io\_statement} \rangle ::= \text{ T\_PRINT T\_LPARENTESE } \langle \text{expression} \rangle \text{ T\_RPARENTESE}$   
 $\text{ T\_SEMICOLON}$   
|  $\text{ T\_READ T\_LPARENTESE T\_ID T\_RPARENTESE T\_SEMICOLON}$

$\langle \text{expression} \rangle ::= \text{ T\_ID}$   
|  $\text{ T\_NUMBER}$   
|  $\text{ T\_TRUE}$   
|  $\text{ T\_FALSE}$

```

| T_STRING
| <expression> T_PLUS <expression>
| <expression> T_MINUS <expression>
| <expression> T_TIMES <expression>
| <expression> T_DIV <expression>
| <expression> T_EQ <expression>
| <expression> T_NE <expression>
| <expression> T_LT <expression>
| <expression> T_LE <expression>
| <expression> T_GT <expression>
| <expression> T_GE <expression>
| <expression> T_AND <expression>
| <expression> T_OR <expression>
| T_NOT <expression>
| T_MINUS <expression>
| T_LPARENTESE <expression> T_RPARENTESE

```

## 2 Decisões de Projeto

- **Resolução do "Dangling Else":** A ambiguidade clássica do "dangling else" (a qual `if` um `else` pertence) foi resolvida utilizando as diretivas de precedência do Bison. Definimos dois "tokens falsos" de precedência: `%nonassoc T_IFX` e `%nonassoc T_ELSE`. A regra `if_statement` que **não** possui `else` foi explicitamente marcada com `%prec T_IFX`. Na prática, isso força o parser, ao encontrar um `T_ELSE` (que gera um conflito `shift/reduce`), a comparar a precedência do token `T_ELSE` com a da regra (definida como `T_IFX`). Como `T_ELSE` foi declarado depois, possui precedência maior, levando o parser a escolher a ação de "shift". Isso faz com que o `else` seja associado ao `if` mais próximo, que é o comportamento desejado para a linguagem.
- **Ambiguidade do Menos Unário:** Para diferenciar o operador de subtração binária (`a - b`) do operador de negação unária (`x = -b`), definimos uma regra de precedência específica, `%precedence T_UMINUS`. Esta regra foi colocada com prioridade maior que a dos operadores multiplicativos (`T_TIMES`, `T_DIV`). Na gramática, a produção `T_MINUS expression` foi marcada com `%prec T_UMINUS`. Isso resolve o conflito `shift/reduce`, garantindo que expressões como `10 * -5` sejam interpretadas como `10 * (-5)` e não como `(10 * -) 5`, o que seria um erro.
- **Precedência e Associatividade de Operadores:** Toda a precedência de operadores foi definida na seção de declarações do Bison. A ordem, de menor para maior precedência, foi: `T_ASSIGN` (associatividade à direita, para `a=b=c`), seguido por `T_OR`, `T_AND`, operadores relacionais (`T_EQ`, `T_NE`, etc.), operadores aditivos (`T_PLUS`, `T_MINUS`), operadores multiplicativos (`T_TIMES`, `T_DIV`) e, por fim, os operadores unários (`T_NOT` e `T_UMINUS`). Todos, exceto a atribuição e os unários, foram definidos com associatividade à esquerda (`%left`).

- **Tratamento de Erros Sintáticos:** O tratamento de erros é centralizado na função `yerror`, que é chamada automaticamente pelo `yyparse()` ao encontrar um token inesperado. A função foi implementada para imprimir no `stderr` a mensagem "Erro Sintático", utilizando as variáveis globais `yylineno` e `token_start_column` mantidas pelo analisador léxico. Isso garante que o usuário seja informado da posição exata onde a estrutura gramatical foi violada.

### 3 Dificuldades Encontradas

- **Relatório Preciso da Coluna de Erro:** A variável `column_number`, por ser um cursor que avança constantemente, sempre armazenava a posição *após* o token ser lido. Isso causava com que um erro em um token que começava na coluna 5, por exemplo, fosse reportado na coluna 10 (posição inicial 5 + `yylen` de 5). Para corrigir isso, foi introduzida a variável `token_start_column`. Em cada regra do Flex que retorna um token, esta nova variável armazena a posição de início (o valor de `column_number` *antes* do incremento). A função `yerror` foi então modificada para reportar `token_start_column`, garantindo que a mensagem de erro aponte para o local exato onde o token inesperado começou.
- **Definição da Gramática (Recursão):** Estruturar as regras recursivas da gramática, como `declaration_list` e `block_statement_list`, exigiu atenção. Garantir que as produções vazias (epsilon) estivessem corretas para permitir programas ou blocos vazios, e ao mesmo tempo evitar ambiguidades ou recursões infinitas, foi um ponto de depuração importante.
- **Configuração de Precedência:** Definir a ordem correta de precedência para mais de uma dúzia de operadores foi um processo complicado. Um erro na ordem (por exemplo, colocar `T_AND` com precedência maior que `T_EQ`) levaria a uma análise incorreta de expressões lógicas comuns.

### 4 Conjuntos FIRST e FOLLOW

- $\text{FIRST}(\text{expression}) = \{ T\_ID, T\_NUMBER, T\_TRUE, T\_FALSE, T\_STRING, T\_NOT, T\_MINUS, T\_LPARENTESE \}$
- $\text{FOLLOW}(\text{expression}) = \{ T\_SEMICOLON, T\_RPARENTESE, T\_PLUS, T\_MINUS, T\_TIMES, T\_DIV, T\_EQ, T\_NE, T\_LT, T\_LE, T\_GT, T\_GE, T\_AND, T\_OR \}$

### 5 Análise de Conflitos do Autômato

#### 5.1 Conflito "Dangling Else"

Ocorre quando um `T_ELSE` é encontrado, e o parser não sabe a qual `T_IF` aninhado ele pertence.

```

1 Estado 86
2
3     20 if_statement: T_IF T_LPARENTESE expression T_RPARENTESE
4         statement .
5
6     21           | T_IF T_LPARENTESE expression T_RPARENTESE
7         statement . T_ELSE statement
8
9     T_ELSE deslocar, e ir ao estado 90
10
11     $padrão reduzir usando a regra 20 (if_statement)

```

O Estado 86 ilustra perfeitamente o conflito "dangling else". O parser acaba de ler um `statement` após um `if`. Se o próximo token for um `T_ELSE`, ele tem duas opções:

1. **Deslocar (shift):** Ir para o estado 90, assumindo que o `T_ELSE` pertence a este `if` (Regra 21).
2. **Reduzir (reduce):** Ignorar o `T_ELSE` e fechar a regra 20 (`if_statement` sem `else`).

As diretivas `%nonassoc T_IFX` e `%nonassoc T_ELSE` resolvem este conflito forçando o parser a sempre escolher a ação "shift". Isso associa o `else` ao `if` mais interno e próximo, que é o comportamento correto e esperado para a linguagem.

## 5.2 Conflito do Menos Unário (-)

O segundo conflito é a ambiguidade do token `T_MINUS`, que pode significar tanto uma subtração (binária) quanto uma negação (unária). Este conflito foi resolvido com diretivas de precedência.

### Apresentação dos Estados 25 e 52:

```

1 Estado 25
2
3     31 expression: expression . T_PLUS expression
4     ...
5     44           | T_MINUS expression .
6
7     $padrão reduzir usando a regra 44 (expression)
8
9     -----
10
11 Estado 52
12
13     31 expression: expression . T_PLUS expression
14     32           | expression . T_MINUS expression
15     32           | expression T_MINUS expression .
16     ...
17     T_TIMES deslocar, e ir ao estado 37

```

18	T_DIV	deslocar , e ir ao estado 38
19		
20	\$padrão	reduzir usando a regra 32 (expression)

Os estados acima provam a resolução da ambiguidade:

- O **Estado 25** é alcançado após o parser ler um T\_MINUS seguido de uma *expression* (Regra 44, o menos unário). A ação padrão é **reduzir** imediatamente. Isso mostra que o menos unário tem uma precedência muito alta, sendo resolvido antes de qualquer operador binário.
- O **Estado 52** é alcançado após ler uma subtração (Regra 32). Aqui, o parser *não* reduz imediatamente. Ele primeiro verifica por operadores de maior precedência (T\_TIMES ou T\_DIV).

Isso foi implementado dando ao token "fantasma" T\_UMINUS uma precedência alta e marcando a regra 44 com %prec T\_UMINUS. Isso garante que o parser sempre dê prioridade à interpretação unária (Estado 25) sobre a binária (Estado 52), resolvendo o conflito.

## 6 Testes e Validação

Para validar o analisador sintático, foi criado um arquivo de teste (`teste_sintatico.bl1`) focado em validar as estruturas gramaticais, a precedência de operadores e a detecção de erros.

### 6.1 Arquivo de Teste Sintático (`teste_sintatico.bl1`)

Este arquivo testa declarações, atribuições com expressões complexas, o "dangling else", laços `while` e inclui um erro sintático deliberado (ausência de ponto-e-vírgula).

```
1  /*
2   * Arquivo de teste para o Analisador SINTATICO
3   * da linguagem "mini C".
4   */
5  int global = 100;
6  bool flag = true;
7
8  int main() {
9      int a;
10     int b = -10; // Teste de menos unario
11
12     // Teste de precedencia de operadores
13     // (5 + 10) * (b / 2) = 15 * -5 = -75
14     a = (5 + 10) * (b / 2);
15
16     // Teste de "Dangling Else"
17     // O 'else' deve pertencer ao 'if(a < 0)'
18     if (flag)
19         if (a < 0)
20             print("a eh negativo!");
21         else
22             print("a nao eh negativo!"); // Este 'else'
23
24     // Teste de laco while e bloco
25     int count = 3;
26     while (count > 0) {
27         print(count);
28         count = count - 1;
29     }
30
31     // Teste de erro sintatico (falta ;)
32     int erro = 50
33
34     print("Fim do teste.");
35 }
```

### 6.2 Saída do Analisador

```
youserz@fedora:~/Documentos/programas/projetoCompiladores/com
~/Documentos/programas/projetoCompiladores/com
(base) youserz@fedora:~/Documentos/programas/projetoCompiladores/com$ bison -d -v -o parser.c parser.y
(base) youserz@fedora:~/Documentos/programas/projetoCompiladores/com$ flex -o lexer.c main.l
(base) youserz@fedora:~/Documentos/programas/projetoCompiladores/com$ gcc -o main parser.c lexer.c -lfl
(base) youserz@fedora:~/Documentos/programas/projetoCompiladores/com$ ./main teste_sintatico.bl1
Erro Sintatico na linha 34, coluna 5: syntax error, unexpected T_PRINT

--- Tabela de Símbolos ---
ID      Lexema      Posição (Linha, Coluna)
-----
1       global      (5, 5)
2       flag        (6, 6)
3       main        (8, 5)
4       a           (9, 9)
5       b           (10, 9)
6       count       (25, 9)
7       erro        (32, 9)
-----
(base) youserz@fedora:~/Documentos/programas/projetoCompiladores/com$
```

Figura 1: Saída gerada pelo analisador sintático.