

Manual do Usuário - Simple C

Felipe Gomes da Silva
Luis Henrique Salomão Lobato

Setembro, 2025

Sumário

1	Introdução	2
2	Testando o Analisador Léxico	2
3	Tokens Reconhecidos	3
4	Descrição da Liguagem Regular - Analisador Léxico	4
5	Linguagem Livre de Contexto - Analisador Sintático	5
5.1	Gramática da Linguagem Livre de Contexto	5
6	Análise de Erros	7
6.1	Tratamento de Erros Léxicos (Flex)	7
6.2	Tratamento de Erros Sintáticos (Bison)	7
6.2.1	Função de Erro Sintático e Rastreamento	8
6.2.2	Precedência e Associatividade	8
7	Gerenciamento de Estados	8
7.1	Comentários	8
7.2	Strings	8
8	Exemplos de uso	9
8.1	Declaração de Variáveis e Tipos de Dados	9
8.2	Estruturas Condicionais e de Controle	9
8.3	Laços de Repetição	9
8.3.1	Exemplo de Laço for	10
8.3.2	Exemplo de Laços while e do-while	10

1 Introdução

Este manual do usuário fornece uma visão geral do Simple C, uma linguagem de programação que visa simplificar o uso de conceitos fundamentais da linguagem C. O Simple C é projetado para ser fácil de aprender e usar, tornando-o ideal para iniciantes em programação, mas removendo algumas funcionalidades avançadas da linguagem C.

Neste momento, abordaremos apenas o analisador léxico da linguagem. Utilizamos a ferramenta Flex para construir o analisador léxico, que é responsável por identificar e classificar os tokens na entrada do código-fonte.

2 Testando o Analisador Léxico

Para testar o analisador léxico do Simple C, siga os passos abaixo:

1. Certifique-se de ter o Flex instalado em seu sistema. Você pode verificar isso executando o comando `flex -version` no terminal.
2. Clone o repositório do Simple C do GitHub:

```
git clone github.com/felipe-gsilva/simple-c
```

3. Navegue até o diretório do projeto:

```
cd simple-c/
```

4. Gere o analisador léxico usando o Flex e o compile com nosso script de build

```
chmod +x build.sh  
./build.sh
```

5. Execute o analisador léxico com um arquivo de entrada que contenha código Simple C:

```
./build/simplec < input_file.c
```

Substitua `input_file.c` pelo caminho do arquivo que você deseja analisar.

6. Teste os analisadores com:

```
chmod +x test.sh  
./test.sh
```

Foram criados alguns arquivos de teste na pasta `tests` do repositório. Você pode usar esses arquivos para verificar o funcionamento do analisador léxico. Cada arquivo de teste contém exemplos de código Simple C que abrangem diferentes aspectos da linguagem, como declarações de variáveis, estruturas de controle, funções, entre outros.

Tenha em vista que o analisador sintático ainda não foi implementado, portanto, mudanças na gramática podem ocorrer futuramente.

3 Tokens Reconhecidos

O analisador léxico da linguagem Simple C toma a decisão de aceitar ou não determinada sentença com base nos tokens destacados na Tabela 1

Tabela 1: Tabela de Tokens do Simple C

Nome do Token	Lexema(s)	Descrição
Palavras-chave: Tipos de Dados		
KEYWORD_INT	int	Palavra-chave para tipo inteiro.
KEYWORD_FLOAT	float	Palavra-chave para tipo ponto flutuante.
KEYWORD_CHAR	char	Palavra-chave para tipo caractere.
KEYWORD_STRING	string	Palavra-chave para tipo string (cadeia de caracteres).
KEYWORD_VOID	void	Palavra-chave para tipo vazio/nulo.
KEYWORD_BOOL	bool	Palavra-chave para tipo booleano.
Palavras-chave: Controle de Fluxo e Comandos		
KEYWORD_IF	if	Inicia uma estrutura condicional.
KEYWORD_ELSE	else	Bloco alternativo de uma estrutura condicional.
KEYWORD_FOR	for	Inicia um laço de repetição 'for'.
KEYWORD_WHILE	while	Inicia um laço de repetição 'while'.
KEYWORD_DO	do	Inicia um laço de repetição 'do-while'.
KEYWORD_SWITCH	switch	Inicia uma estrutura de seleção múltipla.
KEYWORD_CASE	case	Define um rótulo dentro de um 'switch'.
KEYWORD_DEFAULT	default	Define o rótulo padrão de um 'switch'.
KEYWORD_BREAK	break	Interrompe a execução de um laço ou 'switch'.
KEYWORD_CONTINUE	continue	Pula para a próxima iteração de um laço.
KEYWORD_RETURN	return	Retorna um valor de uma função.
Identificadores e Literais		
IDENTIFICADOR	var, _x, ...	Nome de variável, função, etc.
INT	123, 42	Valor literal inteiro.
FLOAT	3.14, 0.5	Valor literal de ponto flutuante.
CHAR	'a', '\n'	Valor literal de caractere.
STRING	"hello"	Valor literal de string.
BOOLEAN_LITERAL	true, false	Valor literal booleano.
Operadores		
OP_SOMA	+	Operador de adição.
OP_SUB	-	Operador de subtração.
OP_MULT	*	Operador de multiplicação.
OP_DIV	/	Operador de divisão.
OP_MOD	%	Operador de módulo (resto da divisão).
OP_ATRIBUICAO	=	Operador de atribuição simples.
OP_INC_ATRIBUICAO	+=	Operador de atribuição com adição.
OP_DEC_ATRIBUICAO	-=	Operador de atribuição com subtração.
OP_MULT_ATRIBUICAO	*=	Operador de atribuição com multiplicação.
OP_DIV_ATRIBUICAO	/=	Operador de atribuição com divisão.
OP_INC	++	Operador de incremento.

Tabela 1: Tabela de Tokens do Simple C (Continuação)

Nome do Token	Lexema(s)	Descrição
OP_DEC	–	Operador de decremento.
OP_IGUAL	==	Operador relacional de igualdade.
OP_DIFERENTE	!=	Operador relacional de desigualdade.
OP_MENOR	<	Operador relacional menor que.
OP_MAIOR	>	Operador relacional maior que.
OP_MENOR_IGUAL	<=	Operador relacional menor ou igual que.
OP_MAIOR_IGUAL	>=	Operador relacional maior ou igual que.
OP_AND	& &	Operador lógico E (AND).
OP_OR		Operador lógico OU (OR).
OP_NOT	!	Operador lógico de negação (NOT).
Pontuadores e Delimitadores		
PONTO_VIRGULA	;	Finalizador de instrução.
DOIS_PONTOS	:	Usado em casos de ‘switch’.
VIRGULA	,	Separador de elementos (ex: em listas).
ABRE_PARENTESES	(Abre lista de parâmetros ou expressão.
FECHA_PARENTESES)	Fecha lista de parâmetros ou expressão.
ABRE_CHAVES	{	Abre um bloco de código.
FECHA_CHAVES	}	Fecha um bloco de código.
ABRE_COLCHETES	[Abre a declaração/acesso de um array.
FECHA_COLCHETES]	Fecha a declaração/acesso de um array.

4 Descrição da Linguagem Regular - Analisador Léxico

A linguagem regular estruturada para a aceitação e classificação das cadeias de caracteres, que compõem o código-fonte da linguagem Simple C, é formalmente descrita através do conjunto de expressões regulares detalhadas na Tabela 2. Esta definição rigorosa não apenas estabelece o vocabulário básico do Simple C (identificadores, literais, operadores e palavras-chave), mas também atua como a primeira linha de defesa contra erros. Através da aplicação precisa dessas regras, é garantido que o processo de reconhecimento de tokens seja robusto e não ambíguo. Por exemplo, o analisador léxico assegura estritamente que variáveis e nomes de funções não possam ser declarados com dígitos como primeiro caractere da cadeia, preservando a sintaxe esperada. Adicionalmente, esta etapa é crucial para a identificação e tratamento de outros erros léxicos críticos, como comentários não fechados ou literais malformados, que, se não detectados, comprometeriam fatalmente as fases subsequentes de análise sintática e semântica.

Tabela 2: Expressões regulares do analisador léxico.

Expressão Regular	Descrição	Tipo
<code>[a-zA-Z_][a-zA-Z0-9_]*</code>	Reconhece identificadores válidos que começam com uma letra ou um sublinhado.	Identificador
<code>0 [+-]?[1-9][0-9]*</code>	Reconhece números inteiros.	Literal
<code>[+-]?([0-9]+\.[0-9]* \.[0-9]+)</code>	Reconhece números de ponto flutuante.	Literal
<code>//.*</code>	Comentário de linha única, que começa com <code>//</code> e vai até o final da linha.	Comentário
<code>/*</code>	Inicia um comentário de bloco.	Comentário
<code><IN_COMMENT> "*/"</code>	Fecha um comentário de bloco.	Comentário
<code>"(\. \.[^"\n])+"</code>	Conteúdo de uma string, incluindo caracteres escapados.	String
<code>'(\. \.[^'\n])+'</code>	Reconhece literais de caracteres.	Caractere
<code>==, !=, <=, >=, etc.</code>	Operadores de comparação, lógicos e aritméticos.	Operador
<code>;, <, >, {, }</code>	Símbolos de pontuação e agrupamento.	Símbolo
<code>{INT}{ID}</code>	Captura identificadores que começam com um número, que é um erro léxico.	Erro
<code>.</code>	Captura qualquer caractere que não corresponda a nenhuma regra.	Erro
<code>[+-]?0[0-9]+</code>	Captura números inteiros com zeros à esquerda, que é um erro léxico.	Erro

5 Linguagem Livre de Contexto - Analisador Sintático

5.1 Gramática da Linguagem Livre de Contexto

A partir da definição dos tokens na Seção 3 com a Tabela 1, a linguagem Simple-C pode ser gerada pela gramática livre de contexto definida pela tupla na Equação 1.

Neste contexto, Σ é o conjunto de variáveis não terminais (alfabeto), P é o conjunto de regras de produção, S é o símbolo inicial da derivação e A é o conjunto de variáveis não

terminais.

$$G = (\Sigma, P, S, A) \quad (1)$$

Conjunto de Símbolos Terminais (Σ) Este é o conjunto de símbolos terminais (tokens) que formam as sentenças da linguagem.

$$\Sigma = \left\{ \begin{array}{lll} \text{IDENTIFICADOR,} & \text{PONTO_VIRGULA,} & \text{OP_ATRIBUICAO,} \\ \text{OP_SOMA,} & \text{OP_SUB,} & \text{OP_MULT,} \\ \text{OP_DIV,} & \text{OP_INC,} & \text{OP_DEC,} \\ \text{KEYWORD_INT,} & \text{KEYWORD_FLOAT,} & \text{KEYWORD_STRING,} \\ \text{KEYWORD_BOOL,} & \text{INT,} & \text{FLOAT,} \\ \text{STRING,} & \text{CHAR,} & \text{BOOLEAN_LITERAL,} \\ \text{ABRE_PARENTESSES,} & \text{FECHA_PARENTESSES,} & \text{ABRE_CHAVES,} \\ \text{FECHA_CHAVES,} & \text{KEYWORD_IF,} & \text{KEYWORD_ELSE,} \\ \text{KEYWORD_FOR,} & \text{KEYWORD_WHILE,} & \text{KEYWORD_DO,} \\ \text{OP_IGUAL,} & \text{OP_DIFERENTE,} & \text{OP_MENOR,} \\ \text{OP_MAIOR,} & \text{OP_MENOR_IGUAL,} & \text{OP_MAIOR_IGUAL,} \\ \varepsilon \end{array} \right\}$$

Conjunto de Regras de Produção (P) : O conjunto das regras de produção é definido por:

$\langle \text{Programa} \rangle \rightarrow \langle \text{ListaDeComandos} \rangle$

$\langle \text{ListaDeComandos} \rangle \rightarrow \langle \text{Comando} \rangle \mid \langle \text{Comando} \rangle \langle \text{ListaDeComandos} \rangle$

$\langle \text{Comando} \rangle \rightarrow \langle \text{Declaracao} \rangle \mid \langle \text{Atribuicao} \rangle \mid \langle \text{EstruturaDeControle} \rangle \mid \langle \text{ChamadaDeFuncao} \rangle \mid \langle \text{Bloco} \rangle$

$\langle \text{Declaracao} \rangle \rightarrow \langle \text{Tipo} \rangle \text{ IDENTIFICADOR } \langle \text{AtribuicaoOpcional} \rangle \text{ PONTO_VIRGULA}$

$\langle \text{Atribuicao} \rangle \rightarrow \text{IDENTIFICADOR } \text{OP_ATRIBUICAO } \langle \text{Expressao} \rangle \text{ PONTO_VIRGULA}$

$\langle \text{AtribuicaoOpcional} \rangle \rightarrow \text{OP_ATRIBUICAO } \langle \text{Expressao} \rangle \mid \varepsilon$

$\langle \text{Tipo} \rangle \rightarrow \text{KEYWORD_INT} \mid \text{KEYWORD_FLOAT} \mid \text{KEYWORD_STRING} \mid \text{KEYWORD_BOOL}$

$\langle \text{Literais} \rangle \rightarrow \text{INT} \mid \text{FLOAT} \mid \text{STRING} \mid \text{CHAR} \mid \text{BOOLEAN_LITERAL}$

$\langle \text{Expressao} \rangle \rightarrow \langle \text{Termo} \rangle \mid \langle \text{Expressao} \rangle \text{OP_SOMA } \langle \text{Termo} \rangle \mid \langle \text{Expressao} \rangle \text{OP_SUB } \langle \text{Termo} \rangle$

$\langle \text{Termo} \rangle \rightarrow \langle \text{Fator} \rangle \mid \langle \text{Termo} \rangle \text{OP_MULT } \langle \text{Fator} \rangle \mid \langle \text{Termo} \rangle \text{OP_DIV } \langle \text{Fator} \rangle$

$\langle \text{Fator} \rangle \rightarrow \text{IDENTIFICADOR} \mid \langle \text{Literais} \rangle \mid \text{ABRE_PARENTESSES } \langle \text{Expressao} \rangle \text{ FECHA_PARENTESSES} \mid \text{OP_INC} \mid \text{OP_DEC}$

$\langle \text{EstruturaDeControle} \rangle \rightarrow \langle \text{If} \rangle \mid \langle \text{For} \rangle \mid \langle \text{While} \rangle$

$\langle \text{If} \rangle \rightarrow \text{KEYWORD_IF } \text{ABRE_PARENTESSES } \langle \text{Condicao} \rangle \text{ FECHA_PARENTESSES } \langle \text{Comando} \rangle \mid \langle \text{ElseOpcional} \rangle$

$\langle \text{ElseOpcional} \rangle \rightarrow \text{KEYWORD_ELSE } \langle \text{Comando} \rangle \mid \varepsilon$

$\langle \text{Condicao} \rangle \rightarrow \langle \text{Expressao} \rangle \langle \text{OperadorRelacional} \rangle \langle \text{Expressao} \rangle$

$\langle \text{OperadorRelacional} \rangle \rightarrow \text{OP_IGUAL} \mid \text{OP_DIFERENTE} \mid \text{OP_MENOR} \mid \text{OP_MAIOR} \mid \text{OP_MENOR_IGUAL} \mid \text{OP_MAIOR_IGUAL}$

$\langle \text{Bloco} \rangle \rightarrow \text{ABRE_CHAVES} \langle \text{ListaDeComandos} \rangle \text{FECHA_CHAVES}$

Conjunto de Não-Terminais (A) O conjunto de símbolos não-terminais é definido por:

$$A = \left\{ \begin{array}{lll} \langle \text{Programa} \rangle, & \langle \text{ListaDeComandos} \rangle, \langle \text{Comando} \rangle, & \\ \langle \text{Declaracao} \rangle, & \langle \text{Atribuicao} \rangle, \langle \text{AtribuicaoOpcional} \rangle, & \\ \langle \text{Tipo} \rangle, & \langle \text{Literais} \rangle, \langle \text{Expressao} \rangle, & \\ \langle \text{Termo} \rangle, & \langle \text{Fator} \rangle, \langle \text{EstruturaDeControle} \rangle, & \\ \langle \text{If} \rangle, & \langle \text{ElseOpcional} \rangle, \langle \text{Condicao} \rangle, & \\ \langle \text{OperadorRelacional} \rangle, & \langle \text{Bloco} \rangle, \langle \text{ChamadaDeFuncao} \rangle, & \\ \langle \text{For} \rangle, & \langle \text{While} \rangle, \langle \text{DoWhile} \rangle & \end{array} \right\}$$

Símbolo Inicial (S) O símbolo inicial da gramática é:

$$S = \langle \text{Programa} \rangle$$

6 Análise de Erros

O Simple C implementa um tratamento de erros em duas fases: léxica (Flex) e sintática (Bison).

6.1 Tratamento de Erros Léxicos (Flex)

O analisador léxico é configurado para detectar e reportar falhas específicas, além de caracteres desconhecidos, garantindo um diagnóstico preciso no código-fonte. Para isto, são utilizadas as variáveis globais `current_line` e `current_col`, que rastreiam a posição exata de cada token reconhecido.

A função de erro léxico, `print_error`, reporta a linha, coluna, o tipo de erro e o texto problemático (`yytext`). Erros capturados incluem:

- Identificadores que iniciam com dígitos.
- Literais de caracteres contendo múltiplos caracteres.
- Construtos não fechados, como comentários de bloco (`/ *`) ou strings (`" . . . "`) no fim do arquivo ou com quebras de linha inesperadas.

6.2 Tratamento de Erros Sintáticos (Bison)

A detecção de erros sintáticos é responsabilidade do Analisador Sintático (Bison). Um erro ocorre quando a sequência de tokens fornecida pelo Flex não pode ser reduzida a nenhuma regra de produção da Gramática Livre de Contexto (G).

6.2.1 Função de Erro Sintático e Rastreamento

Quando um erro de sintaxe é encontrado, o Bison invoca a função `yyerror(...)`. Esta função aproveita as variáveis globais `current_line` e `current_col` (mantidas pelo Flex) para informar ao usuário a linha e coluna exatas onde a violação sintática foi detectada.

A função de erro, definida no arquivo Bison, é:

```
1 void yyerror(const char *s) {
2     fprintf(stderr,
3         "Erro Sintatico na Linha %d, Coluna %d: %s\n",
4         current_line,
5         current_col,
6         s
7     );
8     exit(1);
9 }
```

Código 1: Função para tratamento de erros

6.2.2 Precedência e Associatividade

A ambiguidade sintática, especialmente em expressões complexas, é resolvida pela definição explícita de Precedência e Associatividade dos operadores no arquivo Bison (usando diretivas como `'left'`, `'right'`). Isto garante que a estrutura sintática preferida siga a ordem de avaliação matemática e lógica esperada (ex: multiplicação antes da soma).

7 Gerenciamento de Estados

Para melhor gerenciar o estado do analisador léxico, utilizamos a funcionalidade de estados do Flex. Em especial, foram criados 2 estados diferentes: `IN_COMMENT` e `IN_STRING`.

7.1 Comentários

Os comentários podem aparecer de 2 formas no código: (i) comentários de linha única, que começam com `//` e se estendem até o final da linha; (ii) comentários de bloco, que começam com `/*` e terminam com `*/`.

7.2 Strings

As strings são sequências de caracteres delimitadas por aspas duplas (`"`). O analisador léxico reconhece strings e trata caracteres de escape, como `\n` para nova linha e `\t` para tabulação.

Para isso, foi-se utilizado a função `yytext()` do Flex, que permite concatenar a string lida em `yytext` com a próxima parte da string lida, até que o caractere de fechamento (`"`) seja encontrado.

Com isso, é possível a declaração e atribuição de strings com múltiplas linhas. Segue no Código 2, exemplos diversos sobre este tipo de dado.

```
1 // Teste de literais de string, incluindo escapes
2 string s = "Uma string de teste com \"aspas\" escapadas.";
3 string outra_string_123 = "outra string";
4 string string_com_3_linhas =
5     ""
6     Linha 1\nLinha 2\nLinha 3
7     "";
```

Código 2: Exemplos para o tratamento de strings.

8 Exemplos de uso

Nesta seção serão descritos exemplos do código da Linguagem Simple C, visando introduzir conceitos básicos sobre a sintaxe para o uso da mesma.

8.1 Declaração de Variáveis e Tipos de Dados

Abaixo é apresentado um trecho de código de exemplo da linguagem Simple C, que será processado pelo Analisador Léxico (Seção 3). As cores indicam os tokens reconhecidos, conforme o estilo definido:

```
1      int i = 12345;
2      int i_2 = -12345;
3      float _f = .5; // Ponto flutuante implícito
4      float _f2 = 0.555555;
5      float _f3 = 1.000000;
6      char c = 'A';
7      bool flag = true;
8      char newline = '\n'; // Teste de escape em char
9      int resultado_final = 0;
```

Código 3: Exemplo de Código com Tokens da Simple C

8.2 Estruturas Condicionais e de Controle

Seguem exemplos para estruturas condicionais e de controle. No Código 4, apresenta-se a utilização de um “If-Else”. Já no Código 5, está definida a estrutura “Switch-Case”.

```
1
2      // Estrutura condicional 'if-else' com operadores logicos
3      //e relacionais
4      if (flag == true && (i < 1000 || i >= 2000)) {
5          resultado_final = 1;
6      } else if (!flag) {
7          resultado_final = -1;
8      }
```

Código 4: Exemplo de estrutura de Controle If-Else

```
1      switch (c) {
2          case 'A':
3              i = 1;
4              break;
5          case 'B':
6              i = 2;
7              break;
8          default:
9              i = 0;
10     }
```

Código 5: Exemplo de estrutura de Controle Switch-Case

8.3 Laços de Repetição

A linguagem Simple C suporta três estruturas de repetição fundamentais, que permitem a execução iterativa de blocos de código: o laço for, o laço while e o laço do-while. O

uso dos comandos `break` e `continue` (Tokens listados na Tabela 1) é permitido dentro dos corpos de todos os laços para controlar o fluxo de execução.

8.3.1 Exemplo de Laço `for`

O laço `for` no Simple C segue a estrutura padrão, onde a inicialização, a condição de parada e o passo de iteração são explicitados entre parênteses.

```
1      for (int j = 0; j != 10; j++) {
2          if (j <= 1) {
3              continue; // Pula para a proxima iteracao
4          }
5          if (j > 8) {
6              break; // Interrompe o laço
7          }
8      }
```

Código 6: “Laço de repetição For Loop”

8.3.2 Exemplo de Laços `while` e `do-while`

Os laços condicionais `while` e `do-while` são implementados com base na avaliação de uma expressão relacional. Note que o `do-while` garante pelo menos uma execução do bloco.

```
1      int k = 10;
2      while (k > 5) {
3          k--;
4      }
5
6      do {
7          k++;
8      } while (k < 10);
```

Código 7: “Laços de Repetição While e Do While”