Manual do Usuário - Simple C

Felipe Gomes da Silva
Luis Henrique Salomão Lobato

Setembro, 2025

Sumário

1 Introdução

Este manual do usuário fornece uma visão geral do Simple C, uma linguagem de programação que visa simplicar 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/src/flex/
   ```

4. Gere o analisador léxico usando o Flex e o compile com nosso Makefile:

   ```
   make
   ```

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

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

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

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.
OP_DEC	`-`	Operador de decremento.
OP_IGUAL	`==`	Operador relacional de igualdade.
OP_DIFERENTE	`!=`	Operador relacional de desigualdade.
OP_MENOR	`<`	Operador relacional menor que.

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

Nome do Token	Lexema(s)	Descrição
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 Análise de Erros

A função `print_error` exibe as mensagens de erro léxico e inclui a linha, coluna e token que causaram o erro. Para isto, utilizamos rastreamento de posição durante toda a leitura do código-fonte.

4.1 Rastreamento de Posição

Para tratar os erros léxicos de forma eficiente, o analisador léxico do Simple C implementa um mecanismo de rastreamento de posição. Para isto, são declaradas 2 variáveis globais, `current_line` e `current_col`, que armazenam a linha e a coluna atuais do analisador léxico, respectivamente. Essas variáveis são atualizadas conforme o analisador lê o código-fonte, permitindo que mensagens de erro incluam informações precisas sobre a localização do erro.

5 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`.

5.1 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 `*/`.

5.2 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 `yymore()` 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.

6 Gramática da Linguagem Simple-C

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.

$$\boldsymbol{G = (\Sigma, P, S, A)} \tag{1}$$

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

$$\Sigma = \left\{ \begin{array}{lll} \texttt{IDENTIFICADOR}, & \texttt{PONTO\_VIRGULA}, & \texttt{OP\_ATRIBUICAO}, \\ \texttt{OP\_SOMA}, & \texttt{OP\_SUB}, & \texttt{OP\_MULT}, \\ \texttt{OP\_DIV}, & \texttt{OP\_INC}, & \texttt{OP\_DEC}, \\ \texttt{KEYWORD\_INT}, & \texttt{KEYWORD\_FLOAT}, & \texttt{KEYWORD\_STRING}, \\ \texttt{KEYWORD\_BOOL}, & \texttt{INT}, & \texttt{FLOAT}, \\ \texttt{STRING}, & \texttt{CHAR}, & \texttt{BOOLEAN\_LITERAL}, \\ \texttt{ABRE\_PARENTESES}, & \texttt{FECHA\_PARENTESES}, & \texttt{ABRE\_CHAVES}, \\ \texttt{FECHA\_CHAVES}, & \texttt{KEYWORD\_IF}, & \texttt{KEYWORD\_ELSE}, \\ \texttt{KEYWORD\_FOR}, & \texttt{KEYWORD\_WHILE}, & \texttt{KEYWORD\_DO}, \\ \texttt{OP\_IGUAL}, & \texttt{OP\_DIFERENTE}, & \texttt{OP\_MENOR}, \\ \texttt{OP\_MAIOR}, & \texttt{OP\_MENOR\_IGUAL}, & \texttt{OP\_MAIOR\_IGUAL}, \\ \epsilon & & \end{array} \right\}$$

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

⟨**Programa**⟩ → ⟨ListaDeComandos⟩

⟨**ListaDeComandos**⟩ → ⟨Comando⟩ | ⟨Comando⟩ ⟨ListaDeComandos⟩

⟨**Comando**⟩ → ⟨Declaracao⟩ | ⟨Atribuicao⟩ | ⟨EstruturaDeControle⟩ | ⟨ChamadaDeFuncao⟩ | ⟨Bloco⟩

⟨**Declaracao**⟩ → ⟨Tipo⟩ IDENTIFICADOR ⟨AtribuicaoOpcional⟩ PONTO_VIRGULA

⟨**Atribuicao**⟩ → IDENTIFICADOR OP_ATRIBUICAO ⟨Expressao⟩ PONTO_VIRGULA

⟨**AtribuicaoOpcional**⟩ → OP_ATRIBUICAO ⟨Expressao⟩ | ϵ

⟨**Tipo**⟩ → KEYWORD_INT | KEYWORD_FLOAT | KEYWORD_STRING | KEYWORD_BOOL

⟨**Literais**⟩ → INT | FLOAT | STRING | CHAR | BOOLEAN_LITERAL

⟨**Expressao**⟩ → ⟨Termo⟩ | ⟨Expressao⟩ OP_SOMA ⟨Termo⟩ | ⟨Expressao⟩ OP_SUB ⟨Termo⟩

⟨**Termo**⟩ → ⟨Fator⟩ | ⟨Termo⟩ OP_MULT ⟨Fator⟩ | ⟨Termo⟩ OP_DIV ⟨Fator⟩

⟨**Fator**⟩ → IDENTIFICADOR | ⟨Literais⟩ | ABRE_PARENTESES ⟨Expressao⟩ FECHA_PARENTESES | OP_INC | OP_DEC

⟨**EstruturaDeControle**⟩ → ⟨If⟩ | ⟨For⟩ | ⟨While⟩

⟨**If**⟩ → KEYWORD_IF ABRE_PARENTESES ⟨Condicao⟩ FECHA_PARENTESES ⟨Comando⟩ ⟨ElseOpcional⟩

⟨**ElseOpcional**⟩ → KEYWORD_ELSE ⟨Comando⟩ | ϵ

⟨**Condicao**⟩ → ⟨Expressao⟩ ⟨OperadorRelacional⟩ ⟨Expressao⟩

⟨**OperadorRelacional**⟩ → OP_IGUAL | OP_DIFERENTE | OP_MENOR | OP_MAIOR | OP_MENOR_IGUAL | OP_MAIOR_IGUAL

⟨**Bloco**⟩ → ABRE_CHAVES ⟨ListaDeComandos⟩ 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:

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