Compilador fase 1: Análise léxica e sintática

O objetivo desse trabalho é implementar as fases de análise léxica e sintática de um compilador para linguagem **TINY-C** (baseada na **Linguagem C)**. O compilador para a linguagem **TINY-C** restringe a **Linguagem C** para ter apenas tipos **inteiros (`int`)** e **caractere** (**`char`**), comandos condicionais (**`if`**) e repetição (**`while`**) e não implementa a declaração e chamadas de funções, a exceção se faz para as funções de entrada (**`readint`**) e saída (**`writeint`**).

Na implementação do compilador o analisador léxico deve atender as demandas do analisador sintático, a interação entre o analisador léxico e o analisador sintático se dá por meio da função **`consome()`** (**analisador sintático**) que realizará chamadas à função **`obter_atomo()`** (**analisador léxico**).

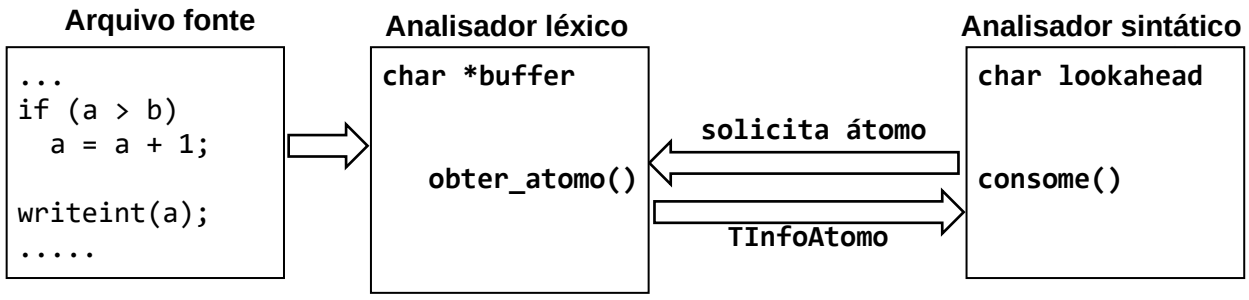

Figura 1: Interação entre Analisador Léxico e Sintático

A seguir são apresentadas a gramática da linguagem **TINY-C**, que deve ser seguida rigorosamente, e as especificações léxicas da linguagem, onde são definidos os átomos da linguagem.

Gramática da linguagem TINY-C

A sintaxe da linguagem **TINY-C** é descrita por uma gramática na notação **EBNF**, vale ressaltar que a notação **EBNF** utiliza os símbolos especiais **|**, **{**, **}**, **[**, **]** **(**, **)** para especificar as regras sintáticas da gramática. Os `<não-terminais>` da gramática são nomes entre parênteses angulares **<** e **>** e os símbolos **`terminais`** (átomos do analisador léxico) estão em **`negrito`** ou entre apóstrofos (Ex: **‘;’**), observe que os símbolos especiais da notação **EBNF** estão em vermelho (ex: **{**, **}**) e os terminais em apóstrofo (ex: **‘{’** e **‘}’**). A construção **{ α }** denotará a repetição da cadeia α zero, uma ou mais vezes **(α*)** e **[β]** é equivalente a **β|λ**, ou seja, indica que a cadeia **β** é opcional.

Considere que o símbolo inicial da gramática é `<program>`:

```
<program> ::= void main ‘(‘ void ‘)’ <compound_stmt>

<compound_stmt> ::= ‘{‘ [ <var_decl> <stmt> ] ‘}’

<var_decl> ::= <type_specifier> <var_decl_list> ‘;’

<type_specifier> ::= int | char

<var_decl_list> ::= <variable_id> { ‘,’ <variable_id> }

<variable_id> ::= id [ ‘=’ <expr> ]

<stmt> ::=  <compound_stmt>          |
            <assig_stmt>             |
            <cond_stmt>              |
            <while_stmt>             |
            readint ‘(‘ id ‘)’ ‘;’   |
            writeint ‘(‘ <expr> ‘)’ ‘;’
```

```
<assig_stmt> ::= id ‘=’ <expr> ‘;’

<cond_stmt> ::= if ‘(‘ <expr> ‘)’ <stmt> [ else <stmt> ]

<while_stmt> ::= while ‘(‘ <expr> ‘)’ <stmt>

<expr> ::= <conjunction> { ‘||’ <conjunction> }

<conjunction> ::= <comparison> { ‘&&’ <comparison> }

<comparison> ::= <sum> [ <relation> <sum> ]

<relation> ::= “<” | “<=” | “==” | “!=” | “>” | “>=”

<sum> ::= <term> { (‘+’ | ‘-’) <term> }

<term> ::= <factor> { ( ‘*’ | ‘/’ ) <factor> }

<factor> ::= intconst | charconst | id | ‘(‘ <expr> ‘)’
```

Especificações Léxicas

- **Caracteres Delimitadores**: Os caracteres delimitadores: espaços em branco, quebra de linhas, tabulação e retorno de carro (‘ ’, ‘\n’, ‘\t’, ‘\r’) deverão ser eliminados (ignorados) pelo analisador léxico, mas o controle de linha (contagem de linha) deverá ser mantido.

- **Comentários**: Existem dois tipos de comentário, um começando com **‘//’** e indo até o final da linha (1 linha) com o finalizador do comentário o caractere **‘\n’**. O outro começando com **“/*”** e terminando com **“*/”** (várias linhas), nesse comentário é importante que a contagem de linha seja mantida.

  Importante: Os comentários devem ser repassados para o analisador sintático para serem reportados e descartados.

- **Identificadores**: Os identificadores começam com uma letra (maiúscula ou minúscula) ou *underline* ‘_’, em seguida pode vir zero ou mais letras (maiúscula ou minúscula) ou *underline* ‘_’ ou dígitos, limitados a 15 caracteres. Caso seja encontrado um identificador com **mais de 15** caracteres deve ser retornado **ERRO** pelo analisador léxico. A seguir a definição regular para **id**.

  letra → a|b|...|z|A|B|...|Z|_
  digito → 0|1|...|9
  id → letra(letra|digito)*

  Importante: Na saída do compilador, para átomo **identificador**, **deverá ser impresso o lexema que gerou o átomo**, ou seja, a sequência de caracteres reconhecida.

- **Palavras reservadas**: As palavras reservadas (em ordem crescente) da linguagem **TINY-C** são: **char**, **else**, **if**, **int**, **main**, **readint**, **void**, **while**, **writeint**.

  Importante: Uma sugestão é que as palavras reservadas sejam reconhecidas na mesma função que reconhece os **identificadores** e deve ser retornado um **átomo específico para cada palavra reservada** reconhecida. Além disso o compilador **é sensível ao caso**, ou seja, **Main** e **main** são átomos diferentes, o primeiro é um **identificador** e o segundo é uma **palavra reservada**.

- **Constantes**:
  Para constante **charconst** são aceitos qualquer caractere da tabela ASCII entre apóstrofo, por exemplo: **'a'** ou **'0'**. Para constante **intconst** o compilador reconhece somente números inteiros na **notação hexadecimal**, conforme a expressão regular abaixo:
  hexa → A|B|C|D|E|F
  intconst → 0x(hexa|digito)+

  Importante: O compilador deve imprimir na tela do computador o caractere que gerou o átomo **charconst** e para átomo **intconst,** deverá ser impresso o **valor numérico na notação decimal do átomo**.

Execução do Compilador

O compilador deve ler o arquivo fonte, com o nome informado por linha de comando, e informar, na tela do computador, a linha e a descrição de todos os átomos reconhecidos no arquivo fonte, o número de linhas analisadas caso o programa esteja sintaticamente correto.

Abaixo temos um exemplo de arquivo fonte em **TINY-C**, sem erros léxicos ou sintáticos, e sua respectiva saída na tela:

Arquivo fonte de entrada.

```
 1 /*
 2 programa le dois numeros inteiros e encontra o maior
 3 */
 4 void main ( void ) {
 5    int num_1, num_2;
 6    int maior;
 7    readint(num_1);
 8    readint(num_2);
 9    if ( num_1 > num_2 )
10        maior = num_1;
11    else
12        maior = num_2;
13
14    writeint(maior); // imprime o maior valor
15 }
```

Saída do compilador na tela

```
#  1:comentario
#  4:void
#  4:main
#  4:abre_par
#  4:void
#  4:fecha_par
#  4:abre_chaves
#  5:int
#  5:id | num_1
#  5:virgula
#  5:id | num_2
#  5:ponto_virgula
#  7:int
..............
15 linhas analisadas, programa sintaticamente correto
```

Caso seja detectado um **erro léxico ou sintático** o compilador deve-se emitir uma mensagem de erro explicativa e terminar a execução do programa. A mensagem explicativa deve informar a linha do erro, o tipo do erro (léxico ou sintático) e caso seja um erro sintático, deve-se informar qual era o **átomo esperado** e qual foi o **átomo encontrado** na análise, veja abaixo um exemplo de saída com erro do compilador

Arquivo fonte de entrada.

```
1  void main ( void ) {
2
3      writeint(maior ;
4  }
```

Exemplo de saída do compilador na tela

```
#  1:void
#  1:main
#  1:abre_par
#  1:void
#  1:fecha_par
#..1:abre_chaves
#  3:writeint
#  3:abre_par
#  3:id | maior
#  3:erro sintatico, esperado [)] encontrado [;]
```

Observações importantes:

O programa deve ser devidamente documentado e poderá ser desenvolvido em grupo de **até dois alunos**. É **imprescindível** que os nomes dos integrantes do grupo sejam mencionados no início do arquivo fonte do trabalho. Além disso, deve-se seguir as **Orientações para Desenvolvimento de Trabalhos Práticos**, as quais estão disponíveis no Moodle.

Fica **terminantemente proibido** o uso de ferramentas de **Inteligência Artificial**, como o ChatGPT, para a **geração automática do código do projeto**. Qualquer tentativa de burlar esta restrição será considerada uma infração disciplinar, conforme o **COMUNICADO DA FCI de 05/02/2025**.

Critérios de Avaliação do Trabalho:

1. **Funcionamento do programa:**
   - Caso o programa não compile ou não execute **será atribuída a nota `0` ao trabalho.**
   - Caso programa apresentarem ***warning*** durante a compilação ou não finalize com retorno igual a `0`, será descontado `1.0` (um ponto) por ***warning*** relatado.
   - O trabalho deve ser desenvolvido na **linguagem C** e será testado usando o compilador do **MinGW** com **VSCode**, para configurar sua máquina no Windows acesse: https://www.doug.dev.br/2022/Instalacoes-e-configuracoes-para-programar-em-C-usando-o-VS-Code/
   - Compile seu programa com o seguinte comando abaixo, considere que o programa fonte do seu compilador seja `compilador.c`:

     ```
     gcc -Wall -Wno-unused-result -g -Og compilador.c -o compilador
     ```

2. **Atendimento à especificação do enunciado:**
   - O quão fiel é o programa quanto à descrição do enunciado, o seu programa deve seguir a gramática definida e realizar a leitura de **programa fonte** armazenado em **arquivo** com o nome informado **por linha de comando.**
   - Clareza e organização, programas com código confuso (linhas longas, variáveis com nomes não-significativos, ...) e desorganizado (sem indentação, sem comentários, ...) também serão penalizados.
     Entrega de um arquivo **Readme.txt** explicando até a parte do trabalho que foi concluído, além de relatar quaisquer *bugs* ou erros identificados na sua implementação. No arquivo você pode compartilhar alguma decisão de *design* e implementação que que foram tomadas durante o desenvolvimento.