Compiladores – 2016/2
Roteiro de Laboratório 2

1 Objetivos

O objetivo deste laboratório é implementar um *scanner* para a linguagem TINY usando o `flex`.

2 A Linguagem TINY

Um programa em TINY tem uma estrutura bastante simples: ele é composto apenas por uma sequência de declarações (*statements*) separadas por ponto-e-vírgula, com uma sintaxe similar a de Ada e Pascal. Não existem funções, procedimentos e declarações. Todas as variáveis são do tipo inteiro, e estas são declaradas através de comandos de atribuição (como em Python). Existem apenas dois comandos de controle: `if` e `repeat`. Ambos podem conter um bloco de comandos. Um comando `if` pode ter uma parte `else` opcional e deve terminar com a palavra chave `end`. Existem também comandos `read` e `write` para realização de operações de entrada e saída básicas. Comentários são escritos entre chaves e não podem ser aninhados.

Expressões em TINY são limitadas a expressões de aritmética de inteiros e de booleanos. Uma expressão booleana consiste de uma comparação entre duas expressões aritméticas usando um dos dois operadores de comparação, `<` e `=`. Uma expressão aritmética pode envolver constantes inteiras, variáveis, parênteses e quaisquer dos quatro operadores de inteiros `+`, `-`, `*` e `/` (divisão inteira), com as propriedades matemáticas usuais. Expressões booleanas poder ser usadas somente como testes em comandos de controle – não existem variáveis booleanas, nem atribuição a variáveis booleanas ou I/O de booleanos.

Apesar da linguagem TINY não possuir muitas das características necessárias a linguagens de programação reais (procedimentos, vetores, e números de ponto flutuante estão entre as omissões mais sérias), a linguagem ainda é suficientemente grande para exemplificar a maioria das características essenciais de um compilador.

Na listagem abaixo temos um exemplo de um programa em TINY para o cálculo da função fatorial.

```
{ Sample program
  in TINY language -
  computes factorial
}
read x; { input an integer }
if 0 < x then { don't compute if x <= 0 }
    fact := 1;
    repeat
        fact := fact * x;
        x := x - 1;
    until x = 0
    write fact; { output factorial of x }
end
```

3 Convenções Léxicas da linguagem TINY

A seção anterior fez apenas uma breve introdução informal da linguagem TINY. A tarefa deste laboratório é construir um *scanner* para a linguagem, e portanto, a sua estrutura léxica deve ser devidamente descrita. Em outras palavras, é necessário definir quais são os tipos de *tokens* e seus respectivos lexemas. Essas informações estão apresentadas na tabela abaixo:

Table 2.1
Tokens of the TINY language

Reserved Words	Special Symbols	Other
if	**+**	***number***
then	**-**	(1 or more
else	*****	digits)
end	**/**	
repeat	**=**	
until	**<**	***identifier***
read	**(**	(1 or more
write	**)**	letters)
	;	
	:=	

Os *tokens* de TINY podem ser classificados em três categorias típicas: palavras reservadas, símbolos especiais e outras. Há oito palavras reservadas, com os significados usuais (embora a sua semântica só será propriamente definida em laboratórios futuros). Existem 10 símbolos especiais, para as quatro operações aritméticas básicas de inteiros, duas operações de comparação (igual e menor que), parênteses, ponto-e-vírgula e atribuição. Todos os símbolos especiais têm comprimento de um caractere, exceto a atribuição, que tem dois caracteres.

Os outros *tokens* são números, que são sequências com um ou mais dígitos, e identificadores, que (para simplificar) são sequências com uma ou mais letras.

Além dos *tokens*, TINY segue as convenções léxicas a seguir. Comentários são cercados por chaves e não podem ser aninhados. O formato do código é livre, isto é, não existem colunas ou posições específicas para uma dada operação. Espaços são formados por espaços em branco, tabulações e quebras de linhas. O princípio de reconhecimento de *tokens* segue um algoritmo guloso: reconhecimento da sub-cadeia de caracteres mais longa.

4 Implementado um *Scanner* para a linguagem TINY

Você deve criar um *scanner* que reconhece todos os elementos léxicos da linguagem TINY. Para tal, utilize o `flex`. Para cada *token* reconhecido no programa de entrada, exiba no terminal:

1. Número da linha do *token* seguido de `:`.
2. Lexema reconhecido seguido de `->`.
3. Tipo do *token* associado ao lexema. Caso o lexema identificado não esteja associado a nenhum tipo de *token*, exiba `UNKNOWN` (desconhecido).

A saída do seu *scanner* para o programa de exemplo anterior fica da seguinte forma:

```
5: read -> READ
5: x -> ID
5: ; -> SEMI
6: if -> IF
6: 0 -> NUM
6: < -> LT
6: x -> ID
6: then -> THEN
7: fact -> ID
7: := -> ASSIGN
7: 1 -> NUM
7: ; -> SEMI
8: repeat -> REPEAT
9: fact -> ID
9: := -> ASSIGN
9: fact -> ID
9: * -> TIMES
9: x -> ID
9: ; -> SEMI
10: x -> ID
10: := -> ASSIGN
10: x -> ID
10: - -> MINUS
10: 1 -> NUM
10: ; -> SEMI
11: until -> UNTIL
11: x -> ID
11: = -> EQ
11: 0 -> NUM
12: write -> WRITE
12: fact -> ID
12: ; -> SEMI
13: end -> END
```

Algumas observações importantes:

- O lexema reconhecido fica guardado na variável global `yytext`.
- O `flex` possui uma variável global para contagem de linha chamada `yylineno`. No entanto, essa variável não é incrementada automaticamente. Você deve fazer isso na sua implementação. Para tal, crie uma regra que reconhece somente o caractere `\n` e incrementa a variável.
- Não esqueça de colocar a regra de reconhecimento de identificadores *depois* das regras de palavras reservadas para evitar sobreposição.
- O tratamento de comentários multi-linha requer algumas considerações especiais, uma vez que não é possível reconhecer uma estrutura casada de abre e fecha chaves somente com uma expressão regular e ainda fazer a contagem de linhas. Nesse caso, há duas alternativas:

1. Ao se reconhecer o caractere '{' coloca-se uma regra que corresponde a uma repetição que consome toda a entrada até que um '}' seja lido. Nesse caso, use a função `input` para ler o próximo caractere.
2. Utilizar regras condicionais conforme explicado no manual do `flex`. Veja, por exemplo, explicações em `http://flex.sourceforge.net/manual/Start-Conditions.html#Start-Conditions`.

- Veja demais exemplos de arquivo de entrada com as respectivas saídas no AVA.

Uma implementação de referência para esse laboratório será disponibilizado pelo professor em um futuro próximo. No entanto, você é *fortemente* encorajado a realizar a sua implementação completa antes de ver uma solução em outro lugar.