

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
INE 5426 - Construção de Compiladores
Relatório

Nome: Fabio Oliveira de Abreu (18100529)
Nome: Bruno Duarte Barreto Borges (18100519)
Nome: Erik Kazuo Sugawara (18100528)

Professor: Alvaro Junio Pereira Franco

1. Identificação dos tokens.

Resposta:

A identificação inicial dos tokens foi retirada da gramática CC-2021-2, onde, para facilitar a organização dentro do código, separamos cada token em cinco grupos: palavras reservadas, operadores, símbolos especiais, constantes e identificadores. Seguem alguns exemplos de tokens com a sua expressão regular.

```
t_ASSIGN = r'\='  
t_GT = r'\>'  
t_LT = r'\<'  
t_EQ = r'\=='  
t_LE = r'\<='  
t_GE = r'\>='  
t_NEQ = r'\!='  
t_PLUS = r'\+ '  
t_MULTIPLY = r'\* '  
t_DIVIDE = r'\/'  
t_REM = r'\%'
```

2. Produção das definições regulares para cada token.

Resposta:

Para produção das definições regulares, foram utilizadas expressões regulares. Como mostrado anteriormente, para definições simples, como os sinais e operadores, foi possível usar a definição em formato de variável. Porém, para definições mais complexas e que necessitavam de tratamento extra, foi necessário usar o formato de função, que também permite o tratamento dos tokens lidos. Segue um exemplo, da definição regular do token "FLOAT" utilizando a ferramenta PLY.

```
def t_float_constant(t):  
    r'[+-]?\d+\.\d+ '  
    t.value = float(t.value)  
    return t
```

3. Construção dos diagramas de transição para cada token.

Resposta:

Os diagramas de transição dos tokens foram feitos em uma tabela que se localiza no arquivo TO-KENS.xlsx. Abaixo, há um exemplo de tabela de transição para um dos tokens identificados (sendo 'digit' a declaração para números de 0 a 9).

int_constant	+	-	digit
→ q0	q1	q1	q2
q1	-	-	q2
*q2	-	-	q2

4. Descrição de uma tabela de símbolos (como foi implementada), quais são os símbolos armazenados na tabela e quais são os atributos dos símbolos escolhidos para armazenar na tabela.

Resposta:

A descrição da tabela de símbolos foi feita através dos tokens que eram retornados pelo analisador léxico, onde, dada uma entrada, a ferramenta identificava seu respectivo token. Com todos tokens identificados, é utilizado o método "print_table" que imprime a tabela de forma elegante no próprio terminal. Os atributos que foram escolhidos para serem mostrados na tabela foram: Token, Valor, Linha e Coluna.

```
def print_table(lexer):
    pattern = "{:~25} | {:~20} | {:~5} | {:~5}"
    print("\033[4m" + pattern.format("TOKEN", "VALUE", "L", "C") + "\033[0m")
    while True:
        tok = lexer.token()
        if not tok:
            break
        print(pattern.format(tok.type, tok.value, tok.lineno, find_column(tok)))

    for e in errors:
        print(e)
```

5. Se não usou ferramenta, uma descrição da implementação do analisador léxico (Usou diagramas de transição? Quais? Quantos? Se não usou diagramas de transição, então o que foi usado?)

Resposta:

Foi utilizado a ferramenta PLY (Python Lex-Yacc).

6. Se usou ferramenta, uma descrição detalhada da entrada exigida pela ferramenta e da saída dada por ela. É necessário haver exemplos pequenos da entrada e da saída gerada pela ferramenta com essa entrada.

Resposta:

Para a criação do analisador léxico, foi utilizada o PLY (Python Lex-Yacc), uma implementação das ferramentas lex e yacc para python. Nela, podemos criar uma lista de tokens a serem identificados pelo analisador, onde devem ser armazenados em uma lista chamada "tokens". As variáveis e funções que se iniciam pelos caracteres "t_" são identificadas como um token pelo interpretador, onde

podemos definir suas respectivas expressões regulares. Executando o PLY em uma string passada, no caso do nosso analisador, obtida de um arquivo cujo caminho é passado como parâmetro 'file' no make, é possível usar o método 'token()' para obter o próximo lexema identificado pelo analisador. O método retorna um objeto com o token, a string identificada, a linha e a posição léxica (que pode ser usada para calcular a coluna). Com isso podemos construir a tabela de símbolos referente ao arquivo passado.

```
$ make run file='tmp/lil_example.lcc'
```

```
( \
python3 main.py tmp/lil_example.lcc; \
)
```

TOKEN	VALUE	LINE	COLUMN
DEF	def	1	1
IDENT	hello_world	1	5
LPAREN	(1	16
RPAREN)	1	17
LBRACE	{	1	19
PRINT	print	2	5
string_constant	"hello world"	2	11
SEMICOLON	;	2	24
RBRACE	}	3	1
INT	int	5	1
IDENT	x	5	5
SEMICOLON	;	5	6
IDENT	x	6	1
ASSIGN	=	6	3
int_constant	10	6	5
SEMICOLON	;	6	7
IF	if	8	1
LPAREN	(8	4
IDENT	x	8	5
GT	>	8	7
int_constant	30	8	9
RPAREN)	8	11
LBRACE	{	8	13
IDENT	{	8	13
IDENT	hello_world	9	5
LPAREN	(9	16
RPAREN)	9	17
SEMICOLON	;	9	18
RBRACE	}	10	1
ELSE	else	10	3
LBRACE	{	10	8
PRINT	print	11	5
string_constant	"erro"	11	11
SEMICOLON	;	11	17
RBRACE	}	12	1