**Análise Léxica e Tabela de Símbolos**

**Descrição do trabalho**

Implementar um analisador léxico para a linguagem PasC.

Seu analisador léxico deverá ser implementado com o auxílio de um autômato finito determinístico. Ele deverá reconhecer um lexema e retornar, a cada chamada, um objeto da classe Token, representando o token reconhecido de acordo com o lexema encontrado.

Para facilitar a implementação, uma Tabela de Símbolos (TS) deverá ser usada. Essa tabela conterá, inicialmente, **todas as palavras reservadas** da linguagem. À medida que novos tokens forem sendo reconhecidos, esses deverão ser consultados na TS antes de serem cadastrados e retornados. Somente palavras reservadas e identificadores serão cadastrados na TS. Não é permitido o cadastro de um mesmo token mais de uma vez na TS.

Resumindo, o Analisador Léxico deverá imprimir a lista de todos os tokens reconhecidos, assim como mostrar o que está cadastrado na Tabela de Símbolos. Na impressão dos tokens, deverá aparecer a tupla <nome, lexema> assim como linha e coluna do token.

Além de reconhecer os tokens da linguagem, seu analisador léxico deverá detectar possíveis erros e reportá-los ao usuário. O programa deverá informar o erro e o local onde ocorreu (linha e coluna), lembrando que em análise léxica tem-se 3 tipos de erros: caracteres desconhecidos (não esperados ou inválidos), string não-fechada antes de quebra de linha e comentário não-fechado antes do fim de arquivo.

Espaços em branco, tabulações, quebras de linhas e comentários não são tokens, ou seja, devem ser descartados/ignorados pelo referido analisador.

Na gramática do PasC, os terminais de um lexema, bem como as palavras reservadas, estão entre aspas duplas para destacá-los, ou seja, **as aspas não são tokens**.

**Faculdade:** Centro universitário de Belo Horizonte (UNIBH) – 2018

**Professor:** Gustavo Fernandes – [gustavo.fernandes@prof.unibh.br](mailto:gustavo.fernandes@prof.unibh.br)

**Autores**: Lucas Magno – 11511738 – [lucasmagnorc@gmail.com](mailto:lucasmagnorc@gmail.com)

Reubert Barbosa – 11517644 – [reubert-unibh@outlook.com](mailto:reubert-unibh@outlook.com)

**Linguagem de programação utilizada**: Python 3.6.5

O analisador léxico contém 4 classes, sendo: **ErrorMessage, InternalFunction, SymbolTable, Tokens.**

Contém **6** arquivos de testes na linguagem **PasC**.

**Detalhamento de classes:**

* **ErrorMessage**

Esta classe recebe um objeto do tipo Error contendo os seguintes atributos: tag, message, row, column. Em seu corpo estão os métodos getters and setters, assim como seu construtor e uma função ‘\_\_str\_\_’.

Sempre que são detectados erros léxicos, são instanciados novos objetos do tipo ErrorMessage com seus respectivos valores.

* **Atributos**

1. Tag: recebe o tipo de erro. Ex: (léxico, sintático).
2. Message: recebe a mensagem do erro. Ex: (caractere desconhecido, caractere inválido).
3. Row: recebe a linha daquele erro.
4. Column: recebe a coluna que se situa o erro.

* **Funções**

1. Getters and Setters: são funções fundamentais para a instância dos atributos e para a recuperação do mesmo.
2. Constructor: função que monta determinado erro.
3. ‘\_\_str\_\_’ tem como principal objetivo nos retornar uma string concatenada com os atributos daquele erro instanciado para que possa ser ‘printado’ o erro para o usuário.

* **Tokens**

Esta classe recebe um objeto do tipo Token contendo os seguintes atributos: tag, value, row, column. Em seu corpo estão os métodos getters and setters, assim como seu construtor e uma função ‘\_\_str\_\_’.

Sempre que são detectados novos tokens na linguagem, são instanciados novos objetos do tipo Token com seus respectivos valores.

* **Atributos**

1. Tag: recebe o tipo de token. Ex: (CON\_NUM, ID, LIT, etc).
2. Value: recebe o valor do token. Ex: (“123”, ‘c’, ‘sVar’, etc).
3. Row: recebe a linha daquele token.
4. Column: recebe a coluna que se situa o token.

* **Funções**

1. Getters and Setters: são funções fundamentais para a instância dos atributos e para a recuperação do mesmo.
2. Constructor: função que monta determinado erro.
3. ‘\_\_str\_\_’ tem como principal objetivo nos retornar uma string concatenada com os atributos daquele token instanciado para que possa ser ‘printado’ para o usuário.

* **SymbolTable**

Esta classe é onde se situa toda a tabela de símbolos. Ela é responsável por guardar todas palavras-chave da linguagem PasC. Ela recebe a importação da classe Tokens.

Nela temos: a variável: symbolTable, e as funções: \_\_init\_\_, addSymbolTable, getToken, \_\_str\_\_

* **Váriavel**

1. symbolTable: váriavel do tipo array, responsável por receber todos as palavras-chave da linguagem PasC assim como os ID’s identificados ao decorrer da leitura do código.

* **Funções**

1. \_\_init\_\_: Função responsável por iniciar as palavras-chave da linguagem PasC, instanciando-as na symbolTable no momento de execução.
2. addSymbolTable: Função responsável por adicionar os símbolos detectados na execução do programa na symbolTable. Ela percorre todos símbolos dentro da symbolTable, verificando se aquele símbolo que deseja adicionar já é existente, senão for existente ela o insere.
3. getToken: Função responsável pela comparação dos tokens encontrados. Temos aqui dois tipos de tokens que vão ser inseridos na symbolTable(tabela de símbolos), sendo palavras-chave da linguagem (KW) e as palavras detectadas no momento de execução do código (ID). No momento de execução, essa função há de comparar se aquele token que ela está recebendo é um ID ou um KW, então ela percorre toda symbolTable comparando se aquele token é um KW (onde todos são previamente instanciados), caso não seja um KW por indução sabemos que é um ID, retornando o token já identificado.
4. \_\_str\_\_: função responsável por imprimir o token situado na symbolTable concatenado com seus atributos.

* InternalFunction

Esta classe é a responsável por dar início a todo processo de reconhecimento léxico. Ela importa todas outras classes previamente citadas.

Temos nela algumas variáveis globais como: Ts (recebe a instância de symbolTable criada na classe SymbolTable), row (linha), column (coluna), lexem (onde vão ser montados os tokens), i ().

* **Funções**

1. openFile: Responsável por abrir o documento que queremos ler e que contém a linguagem PasC escrita. Recebe o nome do arquivo. Gera erro de entrada e saida senão conseguir ler determinado arquivo.
2. structureOptions: Responsável por detectar o caractere recebido e a partir daquele momento fazer a iteração do autômato aplicando as regras para detecção do mesmo. Exemplo: ele recebe o seguinte caractere: ‘\*’, então ele vai para o passo que corresponde e detecta esse caractere como OP\_MUL (operador de multiplicação) e retorna assim o token correspondente. Assim é feito com todos outros detectados: OP\_\*, SMB\_\*, LIT, ID, KW, CON\_\*. Recebe um caracter por iteração. Retorna Token ou Error (caso haja algum).

* Temos a iteração responsável pela leitura de cada caractere no final do corpo do arquivo, lá é feito um for (laço) que percorre linha por linha do arquivo previamente aberto, dentro dele temos um loop infinito (while) que percorre cada caractere daquela linha, passando para a função structureOptions para que sejam identificados os tokens, ao retornar um token, temos uma variável tokenHelper que o recebe e imprime-o, se o token for um ID, adicionamos ele na symbolTable. Temos também uma verificação para imprimir de Errors. Temos a soma do ‘i’ do for de 1 em 1. O loop infinito (while) só termina quando é encontrado o EOF(token que sinaliza end of file(final de arquivo)).

Resumindo, existe um laço para ler cada linha do arquivo e sucessivamente cada caractere, passando para função structureOptions cada caractere detectado, para que seja tomado as medidas cabíveis de acordo com as regras do autômato da linguagem e retornado um token, após isso imprimimos o token e somamos 1 ao ‘i’ (variável de iteração do laço) para que esse laço só tenha fim quando chegarmos ao final do arquivo.

Autômato finito

- O autômato é o mesmo e foi dividido em três imagens diferentes para melhor visualização.





