Tradutores: Analisador Léxico

Guilherme Andreúce Sobreira Monteiro - 14/0141961

Universidade de Brasília - Darcy Ribeiro - CiC/Est, DF/Brasíl
140141961@aluno.unb.br

Abstract. Este trabalho consiste da utilização do programa Flex (gerador léxico) para gerar tokens que serão utilizados como referência para implementar futuramente um tradutor. A línguagem proposta será uma sublinguagem C para lidar com conjuntos. Neste primeiro estágio será apresentado somente como foi feita a geração desses tokens e uma possível gramática que será utilizada para a implementação do tradutor que será construído nos próximos estágios.

Keywords: Tradutor · Flex · Analisador léxico · Tokens.

1 Motivação e Proposta

A línguagem C é uma línguagem muito versátil em termos de construção de estruturas e manipulação de dados, no entanto para que essa versalidade ocorra, o programador precisa entender profundamente o que ele está fazendo. [1] Neste contexto, podemos observar que em C, diferente de Python, a construção das estruturas parte toda do programador; para se construir uma lista em python, basta declarar o tipo da variável, enquanto que em C você tem que construir utilizando estruturas e ponteiros. [2] [3] Assim, para facilitar o uso da línguagem C e, particularmente, suas estruturas, essa sublinguagem surge com essa intenção. Assim será possível utilizar de forma nativa as operações e funcionalidades necessárias para realizar operações com conjuntos de forma simplificada.

2 Analisador léxico

Análise léxica é a primeira fase de um compilador onde este recebe um fluxo de caracteres de um código e os agrupa em tokens. Esses Tokens, também conhecidos como lexemas, são unidades basicas de significado para uma línguagem. Com uma gramática, o analisador léxico é capaz de identificar se esses tokens fazem parte ou não da gramática e se não fazem, onde o erro está localizado. Nesta primeira etapa, o analisador léxico, com o auxílio do programa flex, analisa um trecho de código e separa seus elementos. Para esse em específico, temos:

- ID - Variáveis e funções
- Digito - Números
- OperacoesAritméticas - Token referente a todas operações aritméticas: +, -, *, /

- OperacoesLogicas - Token referente a todas operações lógicas e relacionais: and,or,not,maior,maiorIgual,menor,menorIgual,igual,diferente,atribui,dentroDe
- OperacoesCondicionais - Token referente a todas operações condiconais, de iteração e funções de retorno: if, else, for, forall, exists, return
- OperacoesIO - Token referente a todas operações de entrada e saída: read, write, writeln
- Tipos - Token referente aos tipos: int, float, set, elem, string
- Separadores - Tokens delimitadores: (){},.;

2.1 Funções adicionadas

Para poder realizar a analise de onde existe algum erro léxico, a adição de uma função "print erro" foi adicionada, além de dois parametros para fazer o gerenciamento tanto de linha quanto de caracter, respectivamente, posiçãoLinha e posiçãoCaracter.

2.2 Tratamento de Erros Léxicos

Ao identificar um possível erro léxico, é impresso no terminal o local exato, tanto em posição de caracter quanto a linha em que o programa parou, além de também escrever qual foi o caracter ou token que não pertence a gramática. Facilitando a correção caso necessária.

3 Gramática

Uma primeira possível gramática:

1. *Inicializador* → *global-list*
2. *global-list* → *global-list var* | *var* | *global-list func* | *func* | *global-list coment* | *coment*
3. *coment* → */[*].*[*]/*
4. *var* → *string* **ID**; | *string* **ID = STRING**; | *int* **ID**; | *int* **ID = INT** | *float* **ID**; | *float* **ID = FLOAT** | *set* **ID**; | *set* **ID = SET** | *elem* **ID**; | *elem* **ID = ELEM** | *char* **ID**; | *char* **ID = CHAR**| *char* **ID**; | *set* **ID = SET**
5. *func* → *string* **ID** *(params-list)* { *content-list* } | *int* **ID** *(params-list)* { *content-list* } | **ID** *(params-list)* { *content-list* } | *float* **ID** *(params-list)* { *content-list* } | *set* **ID** *(params-list)* { *content-list* } | *elem* **ID** *(params-list)* { *content-list* }| *char* **ID** *(params-list)* { *content-list* }
6. *params-list* → *params* | ϵ
7. *params* → *params, param* | *param*
8. *param* → *int* **ID** | *float* **ID** | *string* **ID** | *set* **ID** | *elem* **ID** | *char* **ID**
9. *content-list* → *content-list content* | ϵ
10. *content* → *var* | *read* | *write* | *writeln* | *return* | *call-func* | *comand* | *add-value*
11. *add-value* → **ID** = *expression*
12. *expression* → **ID** | **STRING** | **INT** | **DEC** | **SET** | **ELEM** | **CHAR** | **VAZIO** | *expression* **OP** *expression*

13. *comand* → *comand-if* | *comand-ifelse* | *comand-for* | *comand-forall* | *comand-exists* | *comand-add* | *comand-remove*
14. *comand-if* → *if (condition)* {*content-list*}
15. *comand-ifelse* → *if (condition)* { *cotent-list* } *else* {*content-list*}
16. *comand-for* → *for (condition)* {*content-list*}
17. *comand-forall* → *forall (condition)* {*content-list*}
18. *comand-exists* → *exists (condition-set)*
19. *comand-add* → *add (condition-set)*
20. *comand-remove* → *remove (condition-set)*
21. *condition* → **ID COND ID** | **ID COND NUM** | **NUM COND ID** | **NUM COND NUM** | **CHAR COND CHAR**
22. *condition-set* → **ID COND-SET ID** | **ID COND-SET NUM** | **NUM COND-SET ID** | **ID COND-SET VAZIO** |
23. *write* → *write*(**(NUM, ID, NUM)**) | *write*(**(NUM, ID)**| *write*(**(NUM, NUM)**| *write*(**(NUM, STRING)**| *write*(**(NUM, CHAR)**| *write*(**(NUM, VAZIO)**
24. *writeln* → *writeln*(**(NUM, ID, NUM)**) | *writeln*(**(NUM, ID)**| *writeln*(**(NUM, NUM)**| *writeln*(**(NUM, STRING)**| *writeln*(**(NUM, CHAR)**| *writeln*(**(NUM, VAZIO)**
25. *read* → *read*(**(ID, NUM)**) | *read*(**(ID, STRING)** | *read*(**(ID, CHAR)**| *read*(**(ID, VAZIO)**
26. *return* → *return*(**(ID)**) | *return*(**(NUM)** | *return*(**(STRING)**| *return*(**(CHAR)**| *return*(**(VAZIO)**
27. *call-func* → **ID***(call-func-params,)* **ID** | **ID** | ϵ
    - **ID** = *letra (letra | digito | especial)**
    - **INT** = *(digito)+*
    - **DEC** = *(digito)+.(digito)+*
    - **NUM** = **INT** | **DEC**
    - **CHAR** = **'letra'**
    - **STRING** = *(Todos caracteres)**
    - **VAZIO** = *"EMPTY"*
    - **OP** = **+** | **-** | ***** | **/**
    - **COND** = > | >= | < | <= | == | **!=** | **&&** | || | **!** |
    - **COND-SET** = > | >= | < | <= | == | **!=** | **&&** | || | **!** | **in**
    - *letra* = a | ... | z | A | ... | Z
    - *digito* = 0 | ... | 9
    - *especial* = _ | . | ' | "

Palavras reservadas: **read, write, writeln, int, float, string, char, if, else, for, forall, return, set, exists, add, remove, in, empty**
Símbolos reservados: **,|;| (|) | { | }** | " | ' | + | - | * | / | < | > | <= | >= | == | !=

4 Arquivos de teste

O analisador léxico possui 4 testes, onde em 2 desses ele constrói sem erro e no outro ele apresenta o local do erro. Os arquivos são: erro_Léxico1.txt, erro_Léxico2.txt acerto_Léxico1.txt acerto_Léxico2.txt. No caso dos arquivos que apresentam erro, temos o primeiro erro em: Linha 2, Posição 9: Símbolo não permitido. Palavra @; No segundo erro temos na linha 3, posição 16: Símbolo não permitido. Palavra @.

5 Instruções para compilação

Para compilar, tenha instalado o flex (apt get install flex), o gcc (apt get install gcc), vá até a pasta com o arquivo grammar.l, execute "flex grammar.l", gcc lex.yy.c -ll -o guillex e para testar mude de acordo: ./guillex < arquivodeteste.txt > saída.txt Vale lembrar que caso você não aponte o arquivo de entrada, o trabalho pode causar a impressão de que está rodando "infinitamente". Assim, a primeira diretiva de entrada é OBRIGATÓRIA. A diretiva de saída não é necessária, embora facilite a leitura. Observação, por favor rodar o trabalho em ambiente LINUX, de preferência UBUNTU versão 20. Se necessário, instale via máquina virtual!

References

1. A importância e as vantagens de saber programar em linguagem C https://computerworld.com.br/plataformas/importancia-e-vantagens-de-saber-programar-em-linguagem-c/. Acessado em 14 de fevereiro 2020
2. Linked List Program in C https://www.geeksforgeeks.org/linked-list-set-1-introduction/. Acessado em 14 de fevereiro 2020
3. Python lists https://www.w3schools.com/python/python$_{l}ists.asp.Acessadoem14defevereiro2020$
4. Manual Flex https://westes.github.io/flex/manual/. Acessado em 14 de fevereiro 2020