Analisador Léxico

Gabriel Cunha Bessa Vieira - 16/0120811

Universidade de Brasília

1 Introdução

Esse é o relatório inicial do trabalho que será conduzido ao longo do semestre, contando apenas com o analisador léxico dessa vez. O objetivo final com esses trabalhos será construir uma nova linguagem baseada no próprio C. Nas próximas seções serão mostrados a descrição do trabalho e algumas variáveis de ambiente usadas localmente para realizar o trabalho, e por fim será apresentado o esboço de uma gramática seguindo o que foi implementado.

2 Descrição

O intuito inicial do trabalho é realizar uma análise léxica baseada em regex que foram criadas a fim de identificar todos os possíveis tokens que serão usadas nas análises posteriores a essa, sendo elas o analisador sintático, semântico e a geração de código intermediário.

As listas que são propostas para o desenvolvimento desse trabalho são relevantes, pois com elas é possível fazer uma manipulação mais precisa de dados com diferentes tipos, algo que não é possível no C sem a declaração de uma *struct* com diferentes tipos.

Tais tokens são desde caracteres alfanuméricos até operadores aritméticos como '+', '-', '*', '/'. Quando ocorre um comportamento não esperado, é produzido uma linha de erro que aponta no formato '**(linha:coluna) descrição do erro**'.

Para realizar a separação de identificadores e operadores dentro do analisador foram utilizadas macros que têm o nome intuitivo com a sua função.

3 Variáveis de ambiente e versões

3.1 Compilação

Para compilar, é necessário entrar na pasta *src* e rodar o seguinte comando:

```
$ make all
```

Para rodar algum programa é necessário entrar o seguinte comando:

```
$ ./a.out ../tests/<nome_do_programa>.c
```

Os arquivos disponibilizados serão os seguintes:

1. teste_correto1.c
2. teste_correto2.c
3. teste_errado1.c
4. teste_errado2.c

Com o **teste_errado1.c** contendo erro na na linha 2, coluna 10(2:10). E o **teste_errado2.c** contendo erro na linha 2 coluna 9(2:9) e linha 3 coluna 6-8(3:6), (3:7), (3:8) Serão fornecidos dois arquivos teste para cada caso que esteja correto ou errado.

3.2 Software Utilizado

- Ubuntu LTS 20.04
- Flex 2.6.4
- Gcc 11.2.1
- GNU Make 4.3

4 Gramática

1. $< PROGRAM > \rightarrow < DECLARATION\_LIST >$
2. $< DECLARATION\_LIST > \rightarrow < DECLARATION\_LIST >< DECLARATION > | < DECLARATION >$
3. $< DECLARATION > \rightarrow < VAR\_DECLARATION > \ | \ < FUNCTION\_DECLARATION > \ | \ < LIST\_DECLARATION >$
4. $< VAR\_DECLARATION > \rightarrow < SIMPLE\_TYPE >< ID >';' \ | \ < SIMPLE\_TYPE >< ID > [< DIGIT >]';'$
5. $< FUNCTION\_DECLARATION > \rightarrow < TYPE > < ID > \ '(' < PARAMS > \ ')' \ < BRACKETS >$
6. $< LIST\_DECLARATION > \rightarrow < TYPE >< LIST\_TYPE > \ '=' < ID >$
7. $< PARAMS > \rightarrow < TYPE >< ID > \ ','$
8. $< IF\_STATEMENT > \rightarrow \ if \ '(' < EXPRESSION\_STMT > \ ')' \ < BRACKETS >< STATEMENT >< BRACKETS >$
9. $< FOR\_STATEMENT > \rightarrow \ for \ '(' < EXPRESSION\_STMT > \ ')' \ < BRACKETS >< STATEMENT >< BRACKETS >$
10. $< RETURN\_STATEMENT > \rightarrow \ return \ ';' \ | \ return \ < EXPRESSION > \ ';'$
11. $< EXPRESSION\_STMT > \rightarrow < EXPRESSION >';'$
12. $< BRACKETS > \rightarrow \ '\{' \ | \ '\}'$
13. $< KEYWORD > \rightarrow \ if|else|for|return$
14. $< FILTER > \rightarrow \ '<<'$
15. $< MAP > \rightarrow \ '>>'$
16. $< TAIL > \rightarrow \ '!'|'\%'$
17. $< HEADER > \rightarrow \ '?'$
18. $< BINARY\_CONSTRUCTOR > \rightarrow \ ':'$
19. $< OUTPUT > \rightarrow \ write|writeln$

20. $< INPUT > \rightarrow\ read$
21. $< BINARY\_COMP\_OP > \rightarrow\ \ll" \mid \ll = "\| \gg" \mid \gg = " \mid "! = " \mid " == "$
22. $< LOGIC\_OP > \rightarrow\ '\|\|'\mid '\&\&'$
23. $< BINARY\_BASIC\_OP > \rightarrow\ [+ * / -]$
24. $< ID > \rightarrow\ [a - zA - Z\_][a - z0 - 9A - Z\_]*$
25. $< STRING > \rightarrow ("([^{\backslash}"\backslash'])*")$
26. $< SIMPLE\_TYPE > \rightarrow < int > \mid < float >$
27. $< float > \rightarrow < DIGIT > +'.' < DIGIT > +$
28. $< int > \rightarrow < DIGIT > +$
29. $< DIGIT > \rightarrow\ [0 - 9]$
30. $< LIST\_CONSTANT > \rightarrow\ NIL$
31. $< LIST\_TYPE > \rightarrow\ list$

Referências

1. Nalon, C.: Trabalho Prático - Descrição da Linguagem, https://aprender3.unb.br/mod/page/view.php?id=464034
2. Author, F., Author, S.: Title of a proceedings paper. In: Editor, F., Editor, S. (eds.) CONFERENCE 2016, LNCS, vol. 9999, pp. 1–13. Springer, Heidelberg (2016). https://doi.org/10.10007/1234567890
3. A Grammar for the C- Programming Language (Version S21) http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf
4. BNF Grammar for C-Minus http://www.csci-snc.com/ExamplesX/C-Syntax.pdf
5. The syntax of C in Backus-Naur Form https://cs.wmich.edu/ gupta/teaching/cs4850/sumII06/The%20syntax%20of%
Naur%20form.htm