T01 - Analisador Léxico

CIC0106 - Tradutores, Professora Cláudia Nalon, 2020.2

Fillype Alves do Nascimento - Matrícula 16/0070431

UnB - Universidade de Brasília, Brasília - DF, Brasil

1 Motivação

Como parte integrante do processo avaliativo da disciplina de Tradutores do departamento de Ciência da Computação da Universidade de Brasília, foi proposto pela docente a implementação de um compilador. Ao final das entregas, o mesmo deve ser capaz de produzir código intermediário para uma linguagem descrita. A linguagem proposta consiste de um subset da linguagem C acrescido de uma nova primitiva de dados para conjuntos e de operações que permitem trabalhar mais facilmente com esses conjuntos. A linguagem é descrita como se segue.

- **Tipos de dados:** *int* e *float* padrões de C; foram incluídos os novos tipos *set* (conjunto) e *elem*;
- **Constantes:** numéricas para inteiros e reais e a constante *EMPTY* para conjuntos vazios; char e string são usados apenas para impressão;
- **Operadores aritméticos:** $+$, $-$, $*$ e / com semântica e precedência habituais;
- **Operadores lógicos:** !, && e || com semântica e precedência habituais; não existe *boolean*, valores que não sejam 0 ou *EMPTY* são tratados como verdadeiro;
- **Operadores relacionais:** $<$, $<=$, $>$, $>=$, $==$ e $!=$ com semântica e precedência habituais;
- **Escopo:** (, [, {,),] e } para escopo de código; /*, */ e // para escopo de comentários; ' para *char* e *string*;
- **Atribuição:** = para *int* e *float*; por referência para *set*; a depender do tipo da expressão sendo atribuída para *elem*;
- **Controle de fluxo:** *if* e *if-else* para condicionais; *for* para iterações, chamada de função e *return*;
- **I/O:** *read* para entrada de dados, *write* para saída de dados e *writeln* para saída de dados com quebra de linha;
- **Operações sobre conjuntos:**
  - **Pertinência:** o operador binário *in* verifica se um elemento faz parte de um conjunto;
  - **Tipagem:** o operador unário *is_set* verifica se o tipo da variável é *set*;
  - **Inclusão:** o operador *add* é realizado sobre uma operação de pertinência *in* para incluir um membro no conjunto;
  - **Remoção:** o operador *remove* é realizado sobre uma operação de pertinência *in* para remover um membro do conjunto;

– **Seleção:** o operador *exists* é realizado sobre uma operação de pertinência *in* para verificar se um elemento pertence ao conjunto;
– **Iteração:** o operador *forall* é realizado sobre uma operação de pertinência *in* para aplicar alguma expressão a todos os elementos do conjunto.

2 Desenvolvimento

A Análise Léxica é a primeira etapa de compilação. Ela é responsável por separar em tokens o código fonte e apontar erros léxicos. Como ferramenta auxiliar na geração do analisador léxico foi utilizada a linguagem FLEX para descrever as regras a serem consideradas na análise (mais informações sobre o FLEX podem ser encontradas no manual oficial da linguagem [3]). Por meio da descrição de Expressões Regulares, no FLEX, é possível realizar a associação de uma sequência de caracteres presente no código fonte com uma ação a ser executada.

O arquivo ***analex.l*** é o arquivo FLEX que contém a descrição do analisador léxico. Como orientado por Ullman *et. al* em [1], foram criadas declarações de expressões regulares: para cada *keyword* da linguagem; para cada operando, individualmente; um para representar todos os identificadores; para cada constante da linguagem; para cada símbolo de pontuação; para identação; para comentários. As declarações podem ser encontradas no começo do arquivo ***analex.l*** enquanto as regras estão delimitadas entre os caracteres %%. São declaradas variáveis globais para controlar e exibir linha, coluna e contagem de erros léxicos encontrados durante a execução do analisador léxico.

2.1 Funções adicionais

– **main:** função principal do analisador léxico, é responsável por abrir o arquivo texto parâmetro da execução e por imprimir, ao final da análise léxica, a quantidade de erros encontrada durante a análise, se houverem. [2].

```
1   int main(int argc, char *argv[]){
2       yyin = fopen(argv[1], "r");
3       yylex();
4       fclose(yyin);
5
6       if(error_count > 0){
7         printf("\nThe lexical analisys finished with %
              d errors found.\n", error_count);
8       }
9
10      return 0;
11  }
```

3 Arquivos Teste e Anexos

3.1 Arquivos Teste

Foram utilizados oito arquivos de teste. Quatro deles possuem os exemplos disponibilizados pela docente na descrição da linguagem e apresentam exemplos corretos de código, que não geram erros léxicos; estes arquivos estão nomeados por **ex$R.txt** sendo **$** numerado de 1 a 4. Os outros quatro arquivos teste foram compostos originalmente dos mesmos programas dos primeiros quatro arquivos citados (que eram corretos), porém foram introduzidos erros que podem ser captados pelo analisador léxico; estes arquivos estão nomeador por **ex$W.txt** sendo **$** numerado de 1 a 4.

Os erros introduzidos nos arquivos de teste se apresentam como se segue abaixo (detalhados apenas erros léxicos).

- **ex1W.txt:** inserção de caracteres especiais em identificadores de variáveis parâmetros de funções;

```
set 1subsum(set s, int t@rget, int cur_su#m, set
    ans) {
    ...
}
```

- **ex2W.txt:** declaração de identificadores usando apenas caracteres especiais; utilização de aspas duplas para delimitar strings (apenas aspas simples são aceitas);

```
int main() {
    ...
    int %;

    forall (% in s) {
        ...
        forall (val in possibleSums) add((% + val)
            in sumsWithX);
        ...
        if (13 in possibleSums) writeln("y"); else
            writeln("n');
    }
}
```

- **ex3W.txt:** utilização incorreta de delimitadores de comentário multi-linha, levavando o analizador a entender que todo o resto do código é comentário e acabar por ser ignorado;

```
set add_int(set s) {
    ...
}

/* This is a comment block
  acodk
```

```
          apd

        /*
        //

        /

        set add_float(set s) {
            return add(5,4 in add(1. in s));
        }

        ...
        }
```

– **ex4W.txt:** inserção de caracteres especiais em identificadores de funções;

```
    set copy_s&t(set s) {
        ...
    }
    ...
```

3.2 Anexos

Os anexos do primeiro trabalho consistem em:

– **lang_desc:** anotações sobre a descrição da linguagem;
– **grammar:** gramática inicial utilizada (ainda não corrigida);
– **cgrammar:** gramática da linguagem C, utilizada como base para a geração da gramática objeto deste trabalho;
– **analex.l:** arquivo FLEX com as especificações do analisador léxico;
– **lex.yy.c:** arquivo C contendo o analisador léxico, resultado da compilação do arquivo **analex.l**.

4 Instruções de compilação e execução

Para compilar o arquivo flex execute no terminal:

$ flex analex.l

isto irá gerar o programa C com o Analisador Léxico no arquivo *lex.yy.c*; então compile este arquivo usando:

$ gcc lex.yy.c -o analex -lfl

para executar o analisador léxico você deve fornecer para o programa um arquivo *.txt* como input com o código escrito na linguagem especificada:

$ ./analex examples/ex1R.txt

References

1. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers: Principles, Techniques, & Tools. 2nd edn. Pearson, Boston (2007).
2. Flex Manual: Simple Example, `https://westes.github.io/flex/manual/Simple-Examples.html#Simple-Examples`. Last accessed 17 Fev 2021.
3. Flex Manual, `https://westes.github.io/flex/manual/index.html#Top`. Last accessed 17 Fev 2021.
4. Flex (Fast Lexical Analyzer Generator), `https://www.geeksforgeeks.org/flex-fast-lexical-analyzer-generator/`. Last accessed 17 Fev 2021.
5. Writing a simple Compiler on my own - Lexical Analysis using Flex, `https://steemit.com/programming/@drifter1/writing-a-simple-compiler-on-my-own-lexical-analysis-using-flex`. Last accessed 17 Fev 2021.
6. Input and Output Files, `http://web.mit.edu/gnu/doc/html/flex_2.html`. Last accessed 17 Fev 2021.
7. An Overview of flex, with Examples, `http://web.mit.edu/gnu/doc/html/flex_1.html#SEC1`. Last accessed 17 Fev 2021.
8. Flex Regular Expressions, `https://people.cs.aau.dk/~marius/sw/flex/Flex-Regular-Expressions.html`. Last accessed 17 Fev 2021.