Notas de aula - 18/02/2014

Análise Léxica

1. Reconhecer, classificar e interpretar tokens
   (a) "scanner", analisador léxico, "lexer"
   (b) decisões de projeto
      i. strings contêm quebra de linha? Em C, C++, sim (com \); em Pascal não
      ii. Constantes racionais: .1 e 10. são válidas? E se a linguagem tem 1..10? (O contexto só é sabido na gramática)
2. Estrutura típica: função "next_token"
3. implementação manual ou automática
4. Automática:
   (a) Mais fácil
   (b) Mais fácil de mudar
   (c) Eficiência (?)
   (d) Restrita a regexp ("only Perl can parse Perl")
   (e) Não faz interpretação
5. Manual:
   (a) lookahead
   (b) loop
6. Lex
   (a) Baseado em expressões regulares
      i. Strings literais: "foo" ; \caracter ; [a-z] ; [^a-z]...
      ii. União: | (operador infix) ? (operador postfix)
   (b) Sequência, agrupamento entre parênteses
      i. Repetições: 0 ou mais: * (postfix); 1 ou mais: + (postfix)
   (c) Formato:

```
%{
código C
}%
definições
%%
padrão ação
padrão ação
...
%%
código C
```

(d) Ambiguidade:

i. Token mais longo possível

ii. Em caso de empate, usa-se a primeira regra que casou

(e) Exemplos:

i. Constantes double:

```
nat [0-9]+
exp [eE][+-]?{nat}
{nat}{exp}|"."{nat}{exp}?|{nat}"."{nat}?{exp}?
```

ii. Strings: uma versão possível: \"([^"\\]|\\.)*\"

iii. Comentário: "/*" ([^*] | "*" [^/])* "*"? "*/"

(f) String é armazenada na global yytext, de tamanho yyleng

(g) Regra final para capturar erro

7. Análise léxica à mão

(a) Máquina de estados

(b) Lookahead: get_char, put_char, peek_char