Introdução - Analisador Léxico

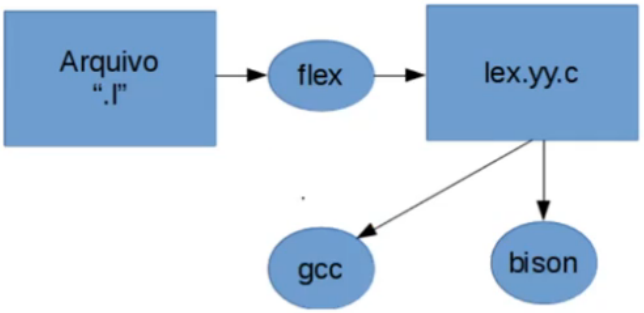
- Compilador
  - Analisador Léxico
  - Analisador Sintático
  - Analisador Semântico
  - Gerador de código
- Analisador léxico
  - Função: isolar palavras-chave, símbolos especiais, etc. transformando-os em códigos mais convenientes para outras fases, por exemplo, analisador sintático.
  - Quando isoladas, essas palavras-chave são chamadas **tokens**
  - Normalmente é mais conveniente referenciá-los com um nome significativo, chamado **símbolo**
  - Exemplo

```
char token[100];
typedef enum simbolos {  simb_program, simb_identificador,
  simb_numero,  simb_abre_parenteses,  simb_virgula, ... };
simbolos simbolo;
```

    token = “program”
    símbolo = simb_program
  - É usado símbolos, para melhor leitura do código
  - O A.L. é um “agente passivo”, chamado sempre que alguém quiser o próximo token da entrada
  - Mantém a posição corrente de leitura
  - Um token é definido com o conjunto de caracteres entre dois separadores: vírgula, ponto, branco, etc.
  - Uma forma “natural” de entendê-lo é implementando uma subrotina com assinatura do tipo:
    Analex(char *token, simbolo *simbolo)
  - Átomo == Token
  - Método 1 - Programa - Autômato Finito Determinístico (complexo)

```
procedimento ANALISADOR_LÉXICO;
início
  átomo:=cadeia_vazia;
  enquanto próximo='□' faça PRÓXIMO;
  se próximo ∈ símbolos_especiais
   então {s:=próximo; PRÓXIMO;
          caso s de
            ':' : se próximo='='
                  então {s:=':=' ; PRÓXIMO};
            '.' : se próximo='.'
                  então {s:='..' ; PRÓXIMO};
            outros: nada
          fim do caso;
          símbolo:=CÓDIGO (s)}
   senão
  se próximo ∈ letras
   então {repita
            átomo:=átomo & próximo; PRÓXIMO
          até próximo ∉ letras_e_dígitos;
          se átomo ∈ palavras_chave
            então símbolo:=CÓDIGO(átomo)
            senão símbolo:=código_de_identificador}
   senão
  se próximo ∈ dígitos
   então {repita
            átomo:=átomo & próximo; PRÓXIMO
          até próximo ∉ dígitos;
          se próximo ∈ letras então ERRO;
          símbolo:=código_de_número}
  senão ERRO
```

- Método 2 - Flex
  - Capaz de reconhecer os tokens criando um autômato finito determinístico para as palavras indicadas
  - A entrada da ferramenta é um arquivo (normalmente extensão .l) com as regras e gera como saída um programa fonte na linguagem C
  - Ao compilar este programa C, o programa executável será um analisador léxico.



  - Um arquivo de entrada do flex é dividido em três partes:
    Definições
    %%
    Regras
    %%
    Subrotinas
  - Definições
    - O que for colocado entre %{...}% aqui será copiado no começo do arquivo lex.yy.c
    - O que vier a seguir são tratados como #define da linguagem C
  - Regras
    - Expressões regulares que devem ser comparados com os caracteres de entrada
    - Assim que encontrar o primeiro match, executa o código associado (entre {...}) e volta ao início
  - Subrotinas
    - Trecho de código que será copiado para o arquivo lex.yy.c
    - Por exemplo, uma subrotina que deve ser executada em várias regras
    - É importante destacar que o arquivo lex.yy.c **não contém um main**, pois ele é normalmente usado junto com o bison