Instituto Tecnológico de Aeronáutica

Laboratório de CES-41

Laboratório III:
Análise Sintática com a ferramenta *Yacc*

Gianluigi Dal Toso

Professor:
Fábio Carneiro Mokarzel

17 de Junho de 2020

Sobre os códigos e arquivos de dados

Anexos à este relatório está um arquivo comprimido em formato `.zip` contendo todos os códigos-fonte desenvolvidos para este laboratório, bem como o arquivo de dados utilizado para testar a implementação de cada tarefa.

O arquivo `lexico.l` contém a implementação do analisador léxico, o arquivo `sintatico.y` contém a implementação do analisador sintático e os arquivos `main.c` e `yyerror.c` servem de apoio para a compilação e execução. O arquivo `sintatico-sem-pretty-print.y` contém uma versão inicial da implementação do analisador sintático, sem a realização do *pretty-print*. A pasta `exemplos/` contém os testes elaborados. Os arquivos com extensão `.comp` representam códigos de entrada e os arquivos com extensão `.dat` referem-se às saídas de execução.

Analisador Léxico

O analisador léxico implementado durante o Laboratório II será utilizado nesta atividade. Algumas alterações no entanto foram realizadas para adaptá-lo ao trabalho em conjunto com o analisador sintático. A saber:

- As definições de tipo do analisador léxico foram transformadas em definições de *tokens* no analisador sintático. Também foi utilizado um `enum` definido no analisado sintático para trabalhar com os atributos dos operandos.
- A definição de `yylval` foi removida do analisado léxico e foi substituita pela definição de uma `union` que realiza este mesmo papel. Os *tokens* que possuem atributos ou propriedades tiveram a definição de tipo inserida ao seu *token*.
- Foi realizada uma correção na expressão regular utilizada para processar os comentários.
- O código da `main` do analisados léxico original foi removida, visto que ela não é mais necessária.

O arquivo `lexico.l` contém a implementação do analisador léxico do laboratório anterior adaptado para esta atividade prática.

Analisador Sintático

O analisador sintático foi implementado baseado nas implementações apresentadas nos slides fornecidos referentes à teoria das atividades práticas. As produções foram retiradas da definição da linguagem **COMP-ITA**.

Ademais, foi implementado um *pretty-print*. As regras para o *pretty-print* foram definidas por minhas próprias preferências estéticas e similaridade com outras linguagens similares. Por exemplo algumas categorias lógicas foram separadas com mais de uma quebra de linha para facilitar a visualização. Como para este laboratório os exemplos de entrada e saída são relativamente extensos, neste relatório será apresentado apenas alguns exemplos curtos de entrada e saída para demonstração do modelo utilizado. Também optei por utilizar espaços ao invés de tabulações na função `tabular()` pois acredito que aumente a "compatibilidade" da saída com diferentes editores de texto.

Análise de arquivos válidos

O Código 1 apresenta um programa válido, porém colocado todo em apenas uma linha. e o Código 2 contém a saída do analisador com *pretty-print* para este caso.

Quadro 1: Arquivo `simple-inline.comp`: Programa válido em apenas uma linha (Aqui mostrado quebrado para fins didáticos).

```
/* Programa para demonstrar a funcionalidade  do analisador sintático implementado */  programa
DemontraLab {{{  var {  int (turma, idade)  carac (nome[20])  carac (inicial) }  funcao int
CalculaMedia ()   var {  int (P1, P2, final) } comandos {  ler (P1);  ler (P2);  final = (P1 + P2) / 2;
retornar final; }   principal  var {  logic (aprovado)  carac (c) }  comandos {  idade = 25; turma =
21; inicial = 'G';  aprovado = falso;   escrever ("Calcular a media? (s/n): ");  ler (c);  se (c == 's'
|| c == 'S') {  se (CalculaMedia() >= 6.5)  aprovado = verdade;  } }  }}}
```

Quadro 2: Arquivo `simple-inline.dat`: Saída obtida no *stdout* para a análise do arquivo `simple-inline.comp`.

```
programa DemontraLab {{{

   var {
      int (turma, idade)
      carac(nome[20])
      carac(inicial)
   }

   funcao int CalculaMedia ()

      var {
         int (P1, P2, final)
      }

      comandos {
         ler (P1);
         ler (P2);
         final = P1 + P2 / 2;
         retornar final;
      }

   principal

      var {
         logic(aprovado)
         carac(c)
      }

      comandos {
         idade = 25;
         turma = 21;
         inicial = 'G';
         aprovado = falso;
         escrever ("Calcular a media? (s/n): ");
         ler (c);
         se (c == 's' || c == 'S') {
            se (CalculaMedia () >= 6.5000)
               aprovado = verdade;
         }
      }

}}}


Programa Compilado com Sucesso!
```

O Código 3 contém um código válido e com um estilo "legível" porém não no formato do *pretty-printer* do analisador e o Código 4 contém a saída do analisador para este código.

Quadro 3: Arquivo `exemplo.comp`: Programa válido porém com estilo diferente do analisador.

```
programa exemplo {{{

var {
        carac (c1, c2, c3, c4[21]) int (i1, i2, i3, i4[42])
    carac (c5, c6) logic (inicio, fim)
}

funcao logic F1(int toso)
var { logic (dummy) }
comandos { retornar dummy; }

funcao int F2()
var { int (dummy) }
comandos { retornar dummy; }

funcao real F3()
var { real (dummy) }
comandos { retornar dummy; }

procedimento Output ()
var {logic(toso)} comandos { escrever("it works!"); }

principal
var {int (i, posic) carac (c) logic (fim)}
comandos {chamar Output ();}

}}}
```

Quadro 4: Arquivo `exemplo.dat`: Saída obtida no *stdout* para a análise do arquivo `exemplo.comp`.

```
1  programa exemplo {{{
2
3     var {
4        carac(c1, c2, c3, c4[21])
5        int (i1, i2, i3, i4[42])
6        carac(c5, c6)
7        logic(inicio, fim)
8     }
9
10    funcao logic F1 (int toso)
11
12       var {
13          logic(dummy)
14       }
15
16       comandos {
17          retornar dummy;
18       }
19
20    funcao int F2 ()
21
22       var {
23          int (dummy)
24       }
```

```
25
26          comandos {
27             retornar dummy;
28          }
29
30       funcao real F3 ()
31
32          var {
33             real (dummy)
34          }
35
36          comandos {
37             retornar dummy;
38          }
39
40       procedimento Output ()
41
42          var {
43             logic(toso)
44          }
45
46          comandos {
47             escrever ("it works!");
48          }
49
50       principal
51
52          var {
53             int (i, posic)
54             carac(c)
55             logic(fim)
56          }
57
58          comandos {
59             chamar Output ();
60          }
61
62    }}}
63
64
65    Programa Compilado com Sucesso!
```

No Código 5 temos a representação do arquivo `completo-inline.comp` que apresenta o código referente ao programa exemplo fornecido com a linguagem em uma versão *inline*. Já o Código 6 representa o arquivo `completo-inline.dat` contendo a saída do analisador sintático para estes arquivos.

Quadro 5: Arquivo `completo-inline.comp`: Código fornecido com a linguagem em versão *inline* (Aqui mostrado quebrado para fins didáticos).

```
/* Programa para contar as ocorrencias das palavras de um texto */ programa |
AnaliseDeTexto {{{ /* Variaveis globais */ var { carac (nomes[50,10], palavra[10])
int (ntab, nocorr[50]) carac (c) logic (fim) } /* Funcao para procurar uma palavra na
tabela de palavras */ funcao int Procura () var { int (i, inf, sup, med, posic,
compara) logic (achou, fimteste) } comandos { achou = falso; inf = 1; sup = ntab;
enquanto (!achou && sup >= inf) { med = (inf + sup) / 2; compara = 0;
fimteste = falso; para i (0; !fimteste && compara == 0; i+1) { se (palavra[i]
< nomes[med,i]) compara = ~1; senao se (palavra[i] > nomes[med,i]) compara =
1; se (palavra[i] == '\0' || nomes[med,i] == '\0') fimteste = verdade; }
```

```
se (compara == 0) achou = verdade; senao se (compara < 0) sup
= med - 1; senao inf = med + 1; } se (achou) posic = med; senao posic = ~inf;
retornar posic; } /* Fim da funcao Procura */ /* Procedimento para inserir uma palavra na
tabela de palavras */ procedimento Inserir (int posic) var {int (i, j) logic (fim)}
comandos { ntab = ntab + 1; para i (ntab; i >= posic+1; i-1) { fim = falso;
para j (0; !fim; j+1) { nomes[i,j] = nomes[i-1,j]; se (nomes[i,j] == '\0') fim =
verdade; } nocorr[i] = nocorr[i-1]; } fim = falso; para j (0; !fim; j+1) {
nomes[posic,j] = palavra[j]; se (palavra[j] == '\0') fim = verdade; } nocorr[posic]
= 1; } /* Fim do procedimento Inserir */ /* Procedimento para escrever a tabela de
palavras */ procedimento ExibirTabela () var {int (i) logic (fim)} comandos { escrever
(" ", "Palavra ", " Num. de ocorr."); para i (1; i <= 50;
i+1) escrever ("-"); para i (1; i <= ntab; i+1) { escrever ("\n "); fim =
falso; para j (0; !fim; j+1) { se (nomes[i,j] == '\0') fim = verdade;
senao escrever (nomes[i,j]); } escrever (" | ", nocorr[i]); } } /* Fim do
procedimento ExibirTabela */ /* Modulo principal */ principal var {int (i, posic) carac
(c) logic (fim)} comandos { ntab = 0; escrever ("Nova palavra? (s/n): "); ler (c);
enquanto (c == 's' || c == 'S') { escrever ("\nDigite a palavra: "); fim = falso;
para i (0; !fim; i+1) { ler (palavra[i]); se (palavra[i] == '\n') { fim =
verdade; palavra[i] = '\0'; } } posic = Procura (); se (posic >
0) nocorr[posic] = nocorr[posic] + 1; senao chamar Inserir (~posic,
i); escrever ("\n\nNova palavra? (s/n): "); ler (c); } chamar ExibirTabela (); } /*
Fim do modulo principal */ }}} /* Fim do programa AnaliseDeTexto */
```

Quadro 6: Arquivo `completo-inline.dat`: Saída obtida no *stdout* para a análise do arquivo `completo-inline.comp`.

```
1  programa AnaliseDeTexto {{{
2
3     var {
4        carac(nomes[50, 10], palavra[10])
5        int (ntab, nocorr[50])
6        carac(c)
7        logic(fim)
8     }
9
10    funcao int Procura ()
11
12       var {
13          int (i, inf, sup, med, posic, compara)
14          logic(achou, fimteste)
15       }
16
17       comandos {
18          achou = falso;
19          inf = 1;
20          sup = ntab;
21          enquanto (!achou && sup >= inf) {
22             med = inf + sup / 2;
23             compara = 0;
24             fimteste = falso;
25             para i (0; !fimteste && compara == 0; i + 1) {
26                se (palavra[i] < nomes[med, i])
27                   compara = ~1;
28                senao
29                   se (palavra[i] > nomes[med, i])
30                      compara = 1;
31                se (palavra[i] == '\0' || nomes[med, i] == '\0')
32                   fimteste = verdade;
```

```
            }
            se (compara == 0)
               achou = verdade;
            senao
               se (compara < 0)
                  sup = med - 1;
               senao
                  inf = med + 1;
         }
         se (achou)
            posic = med;
         senao
            posic = ~inf;
         retornar posic;
      }

   procedimento Inserir (int posic)

      var {
         int (i, j)
         logic(fim)
      }

      comandos {
         ntab = ntab + 1;
         para i (ntab; i >= posic + 1; i - 1) {
            fim = falso;
            para j (0; !fim; j + 1) {
               nomes[i, j] = nomes[i - 1, j];
               se (nomes[i, j] == '\0')
                  fim = verdade;
            }
            nocorr[i] = nocorr[i - 1];
         }
         fim = falso;
         para j (0; !fim; j + 1) {
            nomes[posic, j] = palavra[j];
            se (palavra[j] == '\0')
               fim = verdade;
         }
         nocorr[posic] = 1;
      }

   procedimento ExibirTabela ()

      var {
         int (i)
         logic(fim)
      }

      comandos {
         escrever ("  ", "Palavra  ", "  Num.  de  ocorr.");
         para i (1; i <= 50; i + 1)
            escrever ("-");
         para i (1; i <= ntab; i + 1) {
            escrever ("\n  ");
            fim = falso;
            para j (0; !fim; j + 1) {
               se (nomes[i, j] == '\0')
                  fim = verdade;
               senao
```

```
                escrever (nomes[i, j]);
            }
            escrever ("  |  ", nocorr[i]);
         }
      }

   principal

      var {
         int (i, posic)
         carac(c)
         logic(fim)
      }

      comandos {
         ntab = 0;
         escrever ("Nova  palavra?  (s/n):  ");
         ler (c);
         enquanto (c == 's' || c == 'S') {
            escrever ("\nDigite  a  palavra:  ");
            fim = falso;
            para i (0; !fim; i + 1) {
               ler (palavra[i]);
               se (palavra[i] == '\n') {
                  fim = verdade;
                  palavra[i] = '\0';
               }
            }
            posic = Procura ();
            se (posic > 0)
               nocorr[posic] = nocorr[posic] + 1;
            senao
               chamar Inserir (~posic, i);
            escrever ("\n\nNova  palavra?  (s/n):  ");
            ler (c);
         }
         chamar ExibirTabela ();
      }

}}}


Programa Compilado com Sucesso!
```

Arquivos inválidos sintáticamente

Os Códigos 7, 9, 11 e 13 apresentam código sintaticamente inválidos e os Códigos 8, 10, 12, 14 apresentam as saídas para estes casos.

Quadro 7: Arquivo `erro1.comp`: Arquivo de código sintaticamente inválido.

```
programa Erro1 {{{

   var {
      carac(nomes[50, 10], palavra[10])
      int (ntab, nocorr[50])
      carac(c)
      logic(fim)
```

```
   }

}}}
```

Quadro 8: Arquivo `erro1.dat`: Saída obtida no *stdout* para a análise do arquivo `erro1.comp`.

```
programa Erro1 {{{

   var {
      carac(nomes[50, 10], palavra[10])
      int (ntab, nocorr[50])
      carac(c)
      logic(fim)
   }

syntax error
```

Quadro 9: Arquivo `erro2.comp`: Arquivo de código sintaticamente inválido..

```
programa Erro2 {{{

    principal {}

}}}
```

Quadro 10: Arquivo `erro2.dat`: Saída obtida no *stdout* para a análise do arquivo `erro2.comp`.

```
programa Erro2 {{{

   principal
syntax error
```

Quadro 11: Arquivo `erro3-sem-principal.comp`: Arquivo de código sintaticamente inválido..

```
programa erro3 {{{

var {
        carac (nomes[50,10], palavra[10])
        int (ntab, nocorr[50]) carac (c) logic (fim)
}

funcao int Procura ()

var {
        int (i, inf, sup, med, posic, compara)
            logic (achou, fimteste)
}
comandos {
        achou = falso; inf = 1; sup = ntab;
        enquanto (!achou && sup >= inf) {
                med = (inf + sup) / 2;
```

```
                compara = 0; fimteste = falso;
                para i (0; !fimteste && compara == 0; i+1) {
                    se (palavra[i] < nomes[med,i])
                       compara = ~1;
                    senao se (palavra[i] > nomes[med,i])
                       compara = 1;
                    se (palavra[i] == '\0' || nomes[med,i] == '\0')
                       fimteste = verdade;
                }
                se (compara == 0)
                        achou = verdade;
                senao se (compara < 0)
                        sup = med - 1;
                senao inf = med + 1;
        }
        se (achou) posic = med;
        senao posic = ~inf;
        retornar posic;
}

procedimento Inserir (int posic)

var {int (i, j) logic (fim)}
comandos {
        ntab = ntab + 1;
        para i (ntab; i >= posic+1; i-1) {
        fim = falso;
        para j (0; !fim; j+1) {
            nomes[i,j] = nomes[i-1,j];
            se (nomes[i,j] == '\0') fim = verdade;
        }
           nocorr[i] = nocorr[i-1];
        }
            fim = falso;
            para j (0; !fim; j+1) {
                nomes[posic,j] = palavra[j];
                se (palavra[j] == '\0') fim = verdade;
            }
        nocorr[posic] = 1;

}

procedimento ExibirTabela ()

var {int (i) logic (fim)}
comandos {
        escrever ("          ", "Palavra             ",
                                                          "   Num. de ocorr.");
        para i (1; i <= 50; i+1) escrever ("-");
        para i (1; i <= ntab; i+1) {
                escrever ("\n          "); fim = falso;
                para j (0; !fim; j+1) {
                    se (nomes[i,j] == '\0') fim = verdade;
                    senao escrever (nomes[i,j]);
                   }
                escrever (" | ", nocorr[i]);
        }

}

var {int (i, posic) carac (c) logic (fim)}
```

```
comandos {
        ntab = 0;
        escrever ("Nova palavra? (s/n): ");
        ler (c);
        enquanto (c == 's' || c == 'S') {
                escrever ("\nDigite a palavra: ");
                fim = falso;
                para i (0; !fim; i+1) {
                    ler (palavra[i]);
                    se (palavra[i] == '\n') {
                        fim = verdade;
                        palavra[i] = '\0';
                    }
                }
                posic = Procura ();
                se (posic > 0)
                        nocorr[posic] = nocorr[posic] + 1;
                senao
                        chamar Inserir (~posic, i);
                escrever ("\n\nNova palavra? (s/n): ");
                ler (c);
        }
        chamar ExibirTabela ();



}}} /* Fim do programa AnaliseDeTexto */
```

Quadro 12: Arquivo `erro3-sem-principal.dat`: Saída obtida no *stdout* para a análise do arquivo `erro3-sem-principal.comp`.

```
programa erro3 {{{

   var {
      carac(nomes[50, 10], palavra[10])
      int (ntab, nocorr[50])
      carac(c)
      logic(fim)
   }

   funcao int Procura ()

      var {
         int (i, inf, sup, med, posic, compara)
         logic(achou, fimteste)
      }

      comandos {
         achou = falso;
         inf = 1;
         sup = ntab;
         enquanto (!achou && sup >= inf) {
            med = inf + sup / 2;
            compara = 0;
            fimteste = falso;
            para i (0; !fimteste && compara == 0; i + 1) {
               se (palavra[i] < nomes[med, i])
                  compara = ~1;
```

```
28              senao
29                 se (palavra[i] > nomes[med, i])
30                    compara = 1;
31              se (palavra[i] == '\0' || nomes[med, i] == '\0')
32                 fimteste = verdade;
33           }
34           se (compara == 0)
35              achou = verdade;
36           senao
37              se (compara < 0)
38                 sup = med - 1;
39              senao
40                 inf = med + 1;
41        }
42        se (achou)
43           posic = med;
44        senao
45           posic = ~inf;
46        retornar posic;
47     }
48
49  procedimento Inserir (int posic)
50
51     var {
52        int (i, j)
53        logic(fim)
54     }
55
56     comandos {
57        ntab = ntab + 1;
58        para i (ntab; i >= posic + 1; i - 1) {
59           fim = falso;
60           para j (0; !fim; j + 1) {
61              nomes[i, j] = nomes[i - 1, j];
62              se (nomes[i, j] == '\0')
63                 fim = verdade;
64           }
65           nocorr[i] = nocorr[i - 1];
66        }
67        fim = falso;
68        para j (0; !fim; j + 1) {
69           nomes[posic, j] = palavra[j];
70           se (palavra[j] == '\0')
71              fim = verdade;
72        }
73        nocorr[posic] = 1;
74     }
75
76  procedimento ExibirTabela ()
77
78     var {
79        int (i)
80        logic(fim)
81     }
82
83     comandos {
84        escrever ("          ", "Palavra             ", "   Num. de ocorr.");
85        para i (1; i <= 50; i + 1)
86           escrever ("-");
87        para i (1; i <= ntab; i + 1) {
88           escrever ("\n          ");
```

```
            fim = falso;
            para j (0; !fim; j + 1) {
               se (nomes[i, j] == '\0')
                  fim = verdade;
               senao
                  escrever (nomes[i, j]);
            }
            escrever (" | ", nocorr[i]);
         }
      }

syntax error
```

Quadro 13: Arquivo `erro4-comeco-errado.comp`: Arquivo de código sintaticamente inválido..

```
erro4 {{{

var {
        carac (nomes[50,10], palavra[10])
        int (ntab, nocorr[50]) carac (c) logic (fim)
}

funcao int Procura ()

var {
        int (i, inf, sup, med, posic, compara)
            logic (achou, fimteste)
}
comandos {
        achou = falso; inf = 1; sup = ntab;
        enquanto (!achou && sup >= inf) {
                med = (inf + sup) / 2;
                compara = 0; fimteste = falso;
                para i (0; !fimteste && compara == 0; i+1) {
                    se (palavra[i] < nomes[med,i])
                       compara = ~1;
                    senao se (palavra[i] > nomes[med,i])
                       compara = 1;
                    se (palavra[i] == '\0' || nomes[med,i] == '\0')
                       fimteste = verdade;
                }
                se (compara == 0)
                        achou = verdade;
                senao se (compara < 0)
                        sup = med - 1;
                senao inf = med + 1;
        }
        se (achou) posic = med;
        senao posic = ~inf;
        retornar posic;
}

procedimento Inserir (int posic)

var {int (i, j) logic (fim)}
comandos {
        ntab = ntab + 1;
        para i (ntab; i >= posic+1; i-1) {
```

```
        fim = falso;
        para j (0; !fim; j+1) {
            nomes[i,j] = nomes[i-1,j];
            se (nomes[i,j] == '\0') fim = verdade;
        }
           nocorr[i] = nocorr[i-1];
        }
            fim = falso;
            para j (0; !fim; j+1) {
                nomes[posic,j] = palavra[j];
                se (palavra[j] == '\0') fim = verdade;
            }
        nocorr[posic] = 1;

}

procedimento ExibirTabela ()

var {int (i) logic (fim)}
comandos {
        escrever ("          ", "Palavra             ",
                                                              "   Num. de ocorr.");
        para i (1; i <= 50; i+1) escrever ("-");
        para i (1; i <= ntab; i+1) {
                escrever ("\n          "); fim = falso;
                para j (0; !fim; j+1) {
                    se (nomes[i,j] == '\0') fim = verdade;
                    senao escrever (nomes[i,j]);
                   }
                escrever (" | ", nocorr[i]);
        }

}

principal

var {int (i, posic) carac (c) logic (fim)}
comandos {
        ntab = 0;
        escrever ("Nova palavra? (s/n): ");
        ler (c);
        enquanto (c == 's' || c == 'S') {
                escrever ("\nDigite a palavra: ");
                fim = falso;
                para i (0; !fim; i+1) {
                    ler (palavra[i]);
                    se (palavra[i] == '\n') {
                        fim = verdade;
                        palavra[i] = '\0';
                    }
                }
                posic = Procura ();
                se (posic > 0)
                        nocorr[posic] = nocorr[posic] + 1;
                senao
                        chamar Inserir (~posic, i);
                escrever ("\n\nNova palavra? (s/n): ");
                ler (c);
        }
        chamar ExibirTabela ();

```

```
105
106
107 }}} /* Fim do programa AnaliseDeTexto */
```

Quadro 14: Arquivo `erro4-comeco-errado.dat`: Saída obtida no *stdout* para a análise do arquivo `erro4-comeco-errado.comp`.

```
1 syntax error
```