EA872 Laboratório de Programação de Software Básico
Atividade 2

Vinícius Esperança Mantovani **RA 247395**

Entrega (limite): 16/08/2023, 13:00

Atividade C)

O analisador léxico dado pelo arquivo “p_c.l” funciona como um tradutor de pascal para C, conforme se percebe ao analisar o arquivo. Isso é feito de modo que, na sessão de regras do “.l”, tem-se as seguintes trocas:

1) “.” por espaço em branco
2) “var” por espaço em branco
3) “{” por “/*”
4) “{” por “*/”
5) “or” por “||”
6) “and” por “&&”
7) “begin” por espaço em branco
8) “end” por “}”
9) “if” por “if (”
10) “then” por “)”
11) “program.*[(].*$” por “main()\n{” Onde “.*[(].*$” é uma sequência de caracteres quaisquer, sucedida por um “(” e, por fim, sucedida por outra sequência de caracteres quaisquer e o fim da linha.
12) “[^:><][=]” por “==” Onde “[^:><][=]” é uma sequência composta por um caracter diferente de “:”, “<” e “>”, sucedida por um sinal de igual. Por exemplo “ +”
13) “[:][=]” por “=” Onde “[:][=]” é a sequência “:=”
14) “[<][>]” por “!=” Onde “[<][>]” é a sequência “<>”
15) “^.*integer;” por aquilo que é printado na função “ShuffleInt()” na seção de rotinas do “.l”. Onde “^.*integer;” é uma sequência de caracteres quaisquer, no início de uma linha,sucedida por “integer;”. E, a função “ShuffleInt()” é responsável por printar a sequência “int ” sucedida pela sequência de nomes de inteiros capturadas na variável yytext por meio de um

loop “for” que passa pelos caracteres diferentes de “:” e printa aqueles que os nomes dos inteiros caracter por caracter.

Este comportamento pode ser verificado nas Figuras 1 e 2 a seguir:

```
[pininsu@fedora arquivos_apoio_lex]$ cat input_p_c
program input_p6(input, output);
{ um pequeno programa }
var
 i,j : integer;
  begin
    if (i = 2) then j := 3;
  else
    if (i <= 1) or ((i >= 3) and (i <> 99))
    then j := j mod i;
end.
```

Figura 1: Código em pascal usado como input para a execução de ./p_c

```
[pininsu@fedora arquivos_apoio_lex]$ ./p_c < input_p_c
main()
{
/* um pequeno programa */

int  i,j ;

    if ( (i== 2) ) j = 3;
  else
    if ( (i <= 1) || ((i >= 3) && (i != 99))
    ) j = j % i;
}
```

Figura 2: Código da Figura 1 traduzido de pascal para C executando ./p_c

Atividade D)

O analisador léxico em questão se ocupa de contar e amostrar o número de páginas, linhas, palavras e caracteres do texto passado de entrada, no caso, pelo terminal durante a execução do analisador. Isso pode ser percebido por meio da execução do programa e, também, por meio da análise minuciosa do programa. Portanto, analisemos linha por linha o programa.

Inicialmente, temos, na seção de definições do “.l”, uma linha para declaração os inteiros “lines”, “characs”, “words” e “pages”, todos com o valor , com exceção de “pages”,

cujo valor é definido como 1. Em seguida, temos, ainda nesta seção, a definição do modo de início “Palavra” com “%START Palavra” as definições das macros: “NovaLinha” como “[\n]” (quebra de linha); “Espaco” como “[\t]”; E, “NovaPagina” como “[\f]”. Sabemos que as macros servem para substituirmos uma sequência por outra, como “NovaLinha”, cujas ocorrências são interpretadas como “[\n]”.

Seguindo para a análise da seção de regras, temos o seguinte:

1) Na ocorrência de {Espaco}, ou seja, de “\t”, o analisador é posto para operar no modo padrão pelo comando “BEGIN 0” e soma um no valor da variável “characs”;
2) Na ocorrência de {NovaLinha}, ou seja, de “\n”, o analisador é posto para operar no modo padrão com o mesmo comando do caso anterior e, os valores das variáveis “characs” e “lines” são incrementados em 1;
3) Na ocorrência de {NovaPagina}, ou seja, “\f”, o analisador é posto para operar no modo padrão com o comando do caso anterior e, os valores das variáveis “characs” e “pages” são incrementados em 1;
4) Caso o modo de início seja “Palavra”, então o caractere “.” causa um incremento de 1 na variável “characs”;
5) Por fim, caso o modo de início seja o padrão, então o caractere “.” causa a mudança para o modo “Palavra” e, o incremento de 1 nas variáveis “characs” e “words”.

Por fim, analisemos a seção de código do arquivo. Temos, nesta seção, um loop que dura durante o funcionamento do analisador lógico e, printa os resultados para os números de páginas (“pages”), de linhas (“lines”), de palavras (“words”) e, de caracteres (“characs”).

Assim, notamos o funcionamento do programa por completo, que pode ser melhor visualizado na Figura 3.

```
[pininsu@fedora arquivos_apoio_lex]$ ./p_d input_p_d
olá professor, tudo bem?
este é um teste do analisador p_d
Explicado acima.

Resultados:
1 pagina(s)
3 linha(s)
13 palavra(s)
78 caracter(es)
```

Figura 3: Resposta do terminal para um texto passado ao analisador “p_d”

Atividade E)

Para entendermos o funcionamento do analisador p_e, como fizemos com os anteriores, devemos analisar o arquivo lex “p_e.l”, conforme faz-se a seguir linha por linha:

1) Inicialmente, na seção de definições, temos a definição de um inteiro “valor”
2) Em seguida, ainda nessa seção, temos a definição da macro OCTAL, cujo valor substituído é “[0-7]+O”, ou seja, a expressão regular que indica um número de ao menos um dígito entre 0 e 7 acompanhado pela letra “O”, representando, portanto, um número em base octal.
3) Por fim, na seção de regras temos que, caso o analisador identifique um número octal (sequência definida pela expressão regular citada acima), ele executa os seguintes comandos em C:

   { sscanf(yytext,"%o",&valor); printf("%d",valor); Que converte o valor da sequência registrada pelo analisador em “yytext” em um valor octal (inteiro octal) e, em seguida, printa na saída o valor decimal do octal registrado.
4) Finalmente, no caso de o analisador capturar um “.”, executa-se um “ECHO”, ou seja, imprime-se todo o conteúdo, registrado e alterado até o momento, no output (no caso, no terminal).

Logo, percebe-se que o analisador em questão é responsável por converter os valores em base octal do input em valores na base decimal para o output. Quanto ao resto da sentença, o analisador apenas não altera.

Por fim, podemos observar esse comportamento na prática nas Figuras 4 e 5 abaixo.

```
[pininsu@fedora arquivos_apoio_lex]$ cat input_p_e
O primeiro valor eh 110, seguido de 60, de 60 e de 61.
Agora, temos 600 que equivale a uma certa quantidade. Quanto será?
```

Figura 4: sentença escrita em input_p_e por padrão

```
[pininsu@fedora arquivos_apoio_lex]$ ./p_e < input_p_e
O primeiro valor eh 9, seguido de 6, de 60 e de 61.
Agora, temos 48 que equivale a uma certa quantidade. Quanto será?
```

Figura 5: sentença de input_p_e após atuação do analisador léxico p_e, com octais convertidos para decimais

Atividade F)

O analisador responsável por retirar os comentários de um script shell desenvolvido foi o seguinte:

Unset

```
CommentFREE     ^#\n //captura linhas de comentarios vazios
CommentALL      ^#+[^!\n].*\n //captura linhas de comentarios no
geral
%%

{CommentFREE}   ; //apaga comentarios vazios
```

```
{CommentALL}    ; //apaga comentarios nao vazios
.               ECHO; //copia caracteres que nao sao parte de
sequencias de comentarios

%%
```

E tem como funcionamento detalhado o seguinte:

1) Na primeira linha da seção de definições, definimos a macro “CommentFREE” que é a expressão regular “^#\n”, responsável por capturar linhas vazias de comentário, ou seja, linhas cujo único caractere além da quebra de linha é “#”.
2) Em seguida, ainda nessa seção, temos a definição de “CommentALL”, responsável por capturar as linhas de comentários não vazias, cuja expressão é “^#+[^!\n].*\n”, representante de uma sequência iniciada por um ou mais “#”, no começo de uma linha, sucedido por um caractere diferente de “!” e de “\n”, sucedido por uma sequência de um ou mais caracteres quaisquer, sucedida, finalmente, por um caractere de quebra de linha.
3) Por fim, como regras, apenas são ecoados os caracteres quaisquer e, as sequências que casam com as expressões de comentários são apagadas.

Um exemplo de uso pode ser verificado nas Figuras 6, 7 e 8 abaixo.

```
#!/bin/bash

############################################
# Script que demonstra conceitos do shell  #
############################################

# define a função usage()
usage(){
        echo "Usage: $0 filename"
        exit 1
}

# define a função is_file_exits()
# $f -> guarda o argumento passado ao script
is_file_exits(){
        local f="$1"
        [[ -f "$f" ]] && return 0 || return 1
}
#
#
# chama a função usage
# se não é dado um nome de arquivo
#
[[ $# -eq 0 ]] && usage

# chama is_file_exits
if ( is_file_exits "$1" )
then
 echo "Arquivo encontrado"
else
 echo "Arquivo não encontrado"
fi
```

Figura 6: input de teste para p_f.l (input_p_f)

```
#!/bin/bash


usage(){
        echo "Usage: $0 filename"
        exit 1
}

is_file_exits(){
        local f="$1"
        [[ -f "$f" ]] && return 0 || return 1
}
[[ $# -eq 0 ]] && usage

if ( is_file_exits "$1" )
then
 echo "Arquivo encontrado"
else
 echo "Arquivo não encontrado"
fi
```

Figura 7: result_p_f, saída após remoção dos comentários de input_p_f pelo analisador

```
[pininsu@MiWiFi-RA72-srv arquivos_apoio_lex]$ diff output_p_f result_p_f
[pininsu@MiWiFi-RA72-srv arquivos_apoio_lex]$
```

Figura 8: diff entre result_p_f e output_p_f (não acusa diferença alguma)

Atividade D)

O analisador léxico desenvolvido para escrever datas foi o seguinte:

Unset

```
%{
char *unidades[]={"primeiro","um","dois","tres","quatro","cinco",
          "seis", "sete","oito","nove"};
char *interm[]={"dez","onze","doze","treze",
           "quatorze","quinze","dezesseis",
           "dezessete","dezoito","dezenove"};
char *dezenas[]={"vinte","trinta","quarenta","cinquenta",
           "sessenta","setenta","oitenta","noventa"};
char *meses[]={"janeiro","fevereiro","marco","abril","maio",
          "junho","julho","agosto","setembro",
          "outubro","novembro","dezembro"};
int valor = 0;
int valor2 = 0;
int valor3 = 0;

%}

GeneralFields  [ ]..[/]..[/]..      //expressão regular para capturar a data
```

```
%%
{GeneralFields}    { sscanf(yytext, "%d/%d/%d", &valor, &valor2, &valor3); //separa os
campos da data em tres variaveis diferentes
          printf(" dia ");
          if(valor == 1){ //trata do caso de o dia ser dia primeiro
             printf(unidades[0]);
          } else if(valor < 10){ //trata de dias menores de dez
             printf(unidades[valor]);
          } else if(valor < 20){ //trata de dias menores que vinte e maiores
que 10
             printf(interm[valor - 10]);
          } else if(valor%10 == 0){ // trata de dias vinte e trinta
             printf("%s", dezenas[valor/10 - 2]);
          } else if (valor > 20){ // trata de dias maiores que vinte e
diferentes de trinta
             printf("%s e %s", dezenas[valor/10 - 2], unidades[valor%10]);
          }

          printf(" de ");
          printf("%s", meses[valor2 - 1]); //printa o nome do mes
          printf(" de ");

          if(valor3 <= 30){ //trata do caso de ser anos dois mil
             printf("dois mil");
          } else{ //trata do caso de ser anos mil e novecentos
             printf("mil novecentos");
          }

          if(valor3 == 0){ //trata de o ano ser o ano 0 de algum seculo
          }
          else if(valor3 < 10){ //trata do caso em que o ano esta entre os
nove anos apos o ano 0 do seculo
             printf(" e ");
             printf(unidades[valor3]);
          } else if(valor3 < 20){ //trata da segunda decada de um seculo
             printf(" e ");
```

```
            printf(interm[valor3 - 10]);
        } else if(valor3%10 == 0){ //trata de anos iniciais de decadas
(vinte, trinta...) com excecao do dez
            printf(" e ");
            printf("%s", dezenas[valor3/10 - 2]);
        }
        else if (valor3 > 20){ //trata dos anos maiores que vinte
            printf(" e ");
            printf("%s e %s", dezenas[valor3/10 - 2], unidades[valor3%10]);
        }
    }

. ECHO;

%%
```

Este “.l” funciona da seguinte maneira: há uma expressão regular definida na seção de definições, responsável por capturar ocorrências de datas no formato “xx/xx/xx”, conforme “[ ]..[/]..[/]..”. Dessa maneira, ao capturar uma dessas ocorrências, é usado “sscanf” para se obter cada um dos campos separadamente, cada um em uma das variáveis “valor” (dia), “valor2” (mês) e “valor3” (ano). Em seguida, usando esses valores de campos individualmente, imprime-se “ dia ” no output, seguido pelo número do dia escrito por extenso, então, sucedido por “ de ” e, por fim, o ano escrito por extenso, seguindo o critério de que, para um valor maior que trinta dos dois dígitos representantes do ano na data, temos que o ano é “mil novecentos …”, caso contrário, o ano é “dois mil …”.

Expliquemos então, as condicionais:

1) O primeiro conjunto de condicionais é referente ao dia, determinando, no primeiro caso que seja impresso dia “primeiro”, no segundo, dia “dois” ou “tres” ou … ou “nove”, no terceiro, temos dia “dez”, “onze” ou … ou “dezenove”, no quarto, temos dia “vinte e 1”, “vinte e dois” ou … ou “trinta e um” e, por fim, temos dia “20” ou “30”.

Após isso, ocorre a impressão do mês e, segue-se para o próximo bloco de condicionais;

2) Este conjunto de condicionais é muito parecido com o primeiro, com exceção que não trata diretamente do caso de o valor ser 1, uma vez que o ano não seria “... e primeiro”, mas sim “... e um”. No entanto, uma nova distinção é para os casos em que temos o valor maior que 30 ou menor que 30. No primeiro caso, consideramos que o século é o 20, ou seja “mil novecentos …”, caso contrário, consideramos “dois mil …”.

Um exemplo de funcionamento está exposto a seguir nas Figuras 8 e 9:

```
[pininsu@MiWiFi-RA72-srv arquivos_apoio_lex]$ cat input_p_g
Unicamp foi fundada em  05/10/66.
As comemorações dos 50 anos foram feitas de 29/02/16 a 18/03/16.
Estamos realizando a atividade 2 de EA872.
Houve uma grande preocupação com o funcionamento de sistemas automáticos na vira
da de 31/12/99 para 01/01/00, algo que ficou conhecido como Bug do Milênio.
```

Figura 8: input_p_g por padrão

```
As comemorações dos 50 anos foram feitas de dia vinte e nove de fevereiro de dois
mil e dezesseis a dia dezoito de marco de dois mil e dezesseis.
Estamos realizando a atividade 2 de EA872.
Houve uma grande preocupação com o funcionamento de sistemas automáticos na virada
 de dia trinta e um de dezembro de mil novecentos e noventa e nove para dia primei
ro de janeiro de dois mil, algo que ficou conhecido como Bug do Milênio.
```

Figura 9: input_p_g após aplicação do analisador lógico