Presentacion Beamer

Proyecto 1

Compiladores e Interpretes
Instituto Tecnologico de Costa Rica

2020

Profesor: Dr. Jose Francisco Torres

Examiners:

Nakisha Dixon
2017103353
Andrei Leon 201015265
Francisco Pereira
2017238806

Tabla de contenidos

Flex I

Scanner

FLEX (generador de analizador léxico rápido) es una herramienta para generar analizadores léxicos (escáneres o lexers) escrito por Vern Paxson en C alrededor de 1987. Se usa junto con el generador de parser Berkeley Yacc o el generador de parser GNU Bison

Proceso de Scanning

- Flex genera un archivo .C como salida, 'lex.yy.c' (contiene un montón de código incomprensible pero que básicamente contiene las reglas que especificamos en el incluyendo el código para las acciones que especifiquemos). Este archivo contiene una extern function llamada yylex() que escanea un token.

Proceso de Scanning

- Flex genera un archivo .C como salida, 'lex.yy.c' (contiene un montón de código incomprensible pero que básicamente contiene las reglas que especificamos en el incluyendo el código para las acciones que especifiquemos). Este archivo contiene una extern function llamada yylex() que escanea un token.
- Este archivo se compila y se conecta con la librería 'lfl' para producir un ejecutable. Cuando se corre el ejecutable, analiza cierta entrada buscando ocurrencias de las expresiones regulares.

Proceso de Scanning

- Flex genera un archivo .C como salida, 'lex.yy.c' (contiene un montón de código incomprensible pero que básicamente contiene las reglas que especificamos en el incluyendo el código para las acciones que especifiquemos). Este archivo contiene una extern function llamada yylex() que escanea un token.
- Este archivo se compila y se conecta con la librería 'lfl' para producir un ejecutable. Cuando se corre el ejecutable, analiza cierta entrada buscando ocurrencias de las expresiones regulares.
- Cuando encuentra cierta expresión regular, ejecuta cierto código correspondiente que se le asignó.

Analisis Léxico I

Codigo fuente

A continuación se presenta el código fuente con colores demostrando la división de *tokens*.

Análisis sintáctico por colores I

```
 #include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "Proy1.h"

#define NEWFRM 250
#define HISTOGRAM_LINES 50
#define HSTGRM_SIZE 10
#define DEBUG 0


extern int yylex();



// Line number of read file
```

Análisis sintáctico por colores II

```
extern int yylineno;



// Lexeme of current token


extern char* yytext;

int yylinecounter = NEWFRM;
int yylinelastcount = 1;
int yylinecounter_H = HISTOGRAM_LINES;
int yylinelastcount_H = 1;

char* tokens[TOTALTOKENS] =
{"PREPROCESSOR","COMMENT","KEYWORD","IDENTIFIER","CONSTANTLITERA
char* colors[TOTALTOKENS] =
{"OliveDrab3","Azure4","Turquoise3","Tomato2","HotPink1","Tan2",
```

Análisis sintáctico por colores III

```
char *
commands[]={"gnuplot \"histogram_script.gnu\"","pdflatex main.te

FILE* file;
char* latexFile = "source.tex";


Row getToken(void)
{
      Row myRow;
      int token = yylex();
      myRow.token = token;

      myRow.lexeme = yytext;


// printf("%s\n", myRow.lexeme);
```

Análisis sintáctico por colores IV

```
        return myRow;
}

void addHistogramExtension()
{
        #if DEBUG
        for (int i=0; i<TOTALTOKENS; i++)
                printf(" n:%d ",histogram[current_Hist_i-1].token_cou
        #endif
        fragment_info *tmp;
        current_Hist_s=current_Hist_s+HSTGRM_SIZE;
        tmp = realloc(histogram,
current_Hist_s*sizeof(*histogram));
        if (!tmp)
        {
                printf("failed to realloc");
                exit(9);
```

Análisis sintáctico por colores V

```
//failed to realloc


        }
        else
        {
                #if DEBUG
                printf("extended hist to %d\n",current_Hist_s);
                for (int i=0; i<TOTALTOKENS; i++)
                        printf(" n:%d ",histogram[current_Hist_i].token_
                printf("^ new line of extended hist\n");
                #endif
                histogram = tmp;


//zero out the rows
```

Análisis sintáctico por colores VI

```
                for(int a=0; a<HSTGRM_SIZE; a++)
                        for(int i=i+0; i<TOTALTOKENS; i++)
                                histogram[a+current_Hist_i].token_count[i]=
                #if DEBUG
                printf("extended hist to %d\n",current_Hist_s);
                for (int i=0; i<TOTALTOKENS; i++)
                        printf(" n:%d ",histogram[current_Hist_i].token_
                printf("^ new line of extended hist\n");
                #endif
        }
}

void writeLatexFileStart(FILE* file)
{
        fprintf(file,"\\begin{frame}[fragile,allowframebreaks]{Anál
}
```

Análisis sintáctico por colores VII

```
void writeLatexFileEnd(FILE* file)
{
      fprintf(file,"\n\\end{frame}\n");
}

void writeTokenToLatexFile(Row rowToken,FILE* file)
{
      if(yylineno >= (yylinelastcount + yylinecounter))

// Break beamer frames


      {
            fprintf(file,"\n\\end{frame}\n\\begin{frame}[fragile,
            yylinelastcount+=yylinecounter;
      }
      if(yylineno >= (yylinelastcount_H + yylinecounter_H))

// Histogram array index and realloc
```

Análisis sintáctico por colores VIII

```
        {
                current_Hist_i++;
                #if DEBUG
                printf("hist index ++:%d, yylineno:%d\n",current_Hist
                #endif
                if(current_Hist_i % HSTGRM_SIZE == 0)

// Histogram index is multiple of HSGRM_SIZE (10)


                {
                        addHistogramExtension();
                }
                yylinelastcount_H = yylinelastcount_H +
yylinecounter_H;
        }
        if(rowToken.token==BLANK)
        {
```

Análisis sintáctico por colores IX

```
            if(rowToken.lexeme[0]==0x9)
                  fprintf(file,"\\tab");
            if(rowToken.lexeme[0]==0xa)
                  fprintf(file,"\\newline");
            if(rowToken.lexeme[0]==' ')
                  fprintf(file," ");
      }
      if(rowToken.token==COMMENT)
      {
            fprintf(file,"\\color{%s}\\begin{verbatim}%s\\end{ver
colors[rowToken.token-1], rowToken.lexeme);
      }
      if(rowToken.token!=COMMENT&&rowToken.token!=BLANK)
      {
            fprintf(file,"\\color{%s}\\verb
```

Histograma I

Histograma

A continuación se presenta un histograma que muestra la cantidad de cada tipo de *token* encontrado en el código fuente:

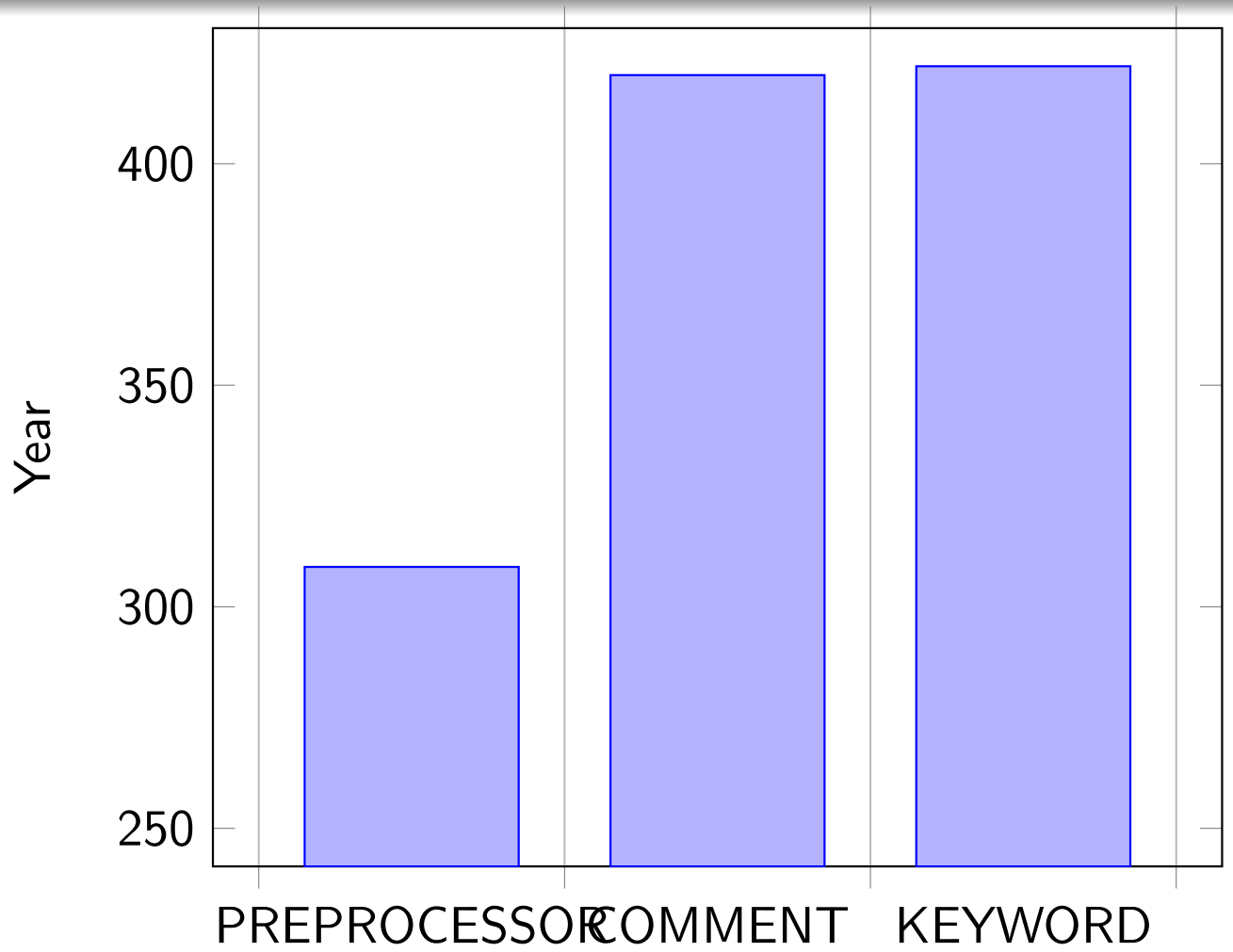
Year
400
350
300
250
PREPROCESSOR
COMMENT
KEYWORD

Grafico Pie I

Pie Chart

A continuación se presenta un grafico de pie que muestra la cantidad de cada tipo de *token* encontrado en el código fuente: