Universidad de San Carlos de Guatemala
Facultad de Ingeniería

Curso: **Organización de Lenguajes y Compiladores 1**

Documentación Técnica: 1er Proyecto

Erik Vladimir Girón Márquez
Carnet # 200313492
Sección C

Guatemala, 21 de Marzo de 2006

Introducción.

Strim (String Manipulator) es un intérprete para operaciones de cadenas de caracteres, implementando análisis léxico y sintáctico sobre el código escrito por el usuario y al mismo tiempo generando los resultados para las operaciones especificadas en el código.

El programa implementa un diseño procedimental en lo que refiere al analizador y a las acciones semánticas, conjugado con un diseño orientado a objetos de C++ en el menú principal de la aplicación. El frontend, utilizando las características de programación por eventos de Visual Basic, simplificaron en gran parte el desarrollo visual e interactivo de la aplicación.

Se utilizó entonces para el desarrollo del proyecto, la plataforma GNU/Linux con las herramientas Flex para generar el analizador léxico, y YACC/Bison para generar parsers LALR; con salida para código fuente compatible con compiladores que cumplan con la norma ANSI C actual. Éste fue compilado tanto para Linux como para Windows utilizando GCC (4.0 y 3.4-mingw respectivamente)

Además de la interfaz de Línea de Comando de la aplicación se implementó un *frontend* para Windows escrito en Visual Basic v. 6, para una manipulación más sencilla de la aplicación bajo un ambiente gráfico en Windows.

Este documento permitirá al programador revisar las expresiones regulares y la gramática libre de contexto utilizada para analizar el texto de entrada. Obviamente, al distribuirse esta aplicación bajo las cláusulas de la licencia GPL v2, podrá revisar mas claramente los archivos fuente que componen el proyecto.

Componentes Léxicos:

A continuación se presenta un resumen de las expresiones regulares utilizadas para reconocer patrones de texto en el proyecto:

```
LETRA [a-zA-Z]
DIGITO [0-9]

{LETRA}({LETRA}|{DIGITO}|\ |\t|\n|\r)*({LETRA}|{DIGITO})*{
return CADENA;} /*regexp para cadenas */;
{DIGITO}+         return NUMERO; /*regexp para numeros */;
"<"              return OP_SHIFTL; /*regexp para operador < */;
">"              return OP_SHIFTR; /*regexp para operador > */;
"<="           return OP_SHIFTNL; /*regexp para operador <= */;
">="           return OP_SHIFTNR; /*regexp para operador >= */;
"\."              return OP_CONCAT; /*regexp para operador  */;
"\^"              return OP_NCONCAT; /*regexp para operador ^ */;
"\("              return LPAREN; /*regexp para ( (agrupacion)*/;
"\)"              return RPAREN; /*regexp para ) (agrupacion) */;
"#"              return END; /*regexp para # (fin de instruccion) */ ;
[\n\r]+                 /* ignorar end of line */;
[\ \t]+                  /* ignorar espacio */;
```

Definición de la Gramática:

En esta sección se presenta las reglas gramaticales utilizadas para analizar sintácticamente la entrada de texto, incluyendo los símbolos terminales y no terminales en orden de precedencia.

```
Terminales con las reglas donde aparecen

$end (0) 0
error (256) 5
OP_CONCAT (258) 12
OP_NCONCAT (259) 13
OP_SHIFTNR (260) 15
OP_SHIFTNL (261) 14
OP_SHIFTR (262) 10
OP_SHIFTL (263) 9
LPAREN (264) 11
RPAREN (265) 11
END (266) 3 4 5
CADENA (267) 6
NUMERO (268) 13 14 15

No terminales con las reglas donde aparecen

$accept (15)
    en la izquierda: 0
instruccion (16)
    en la izquierda: 1 2, en la derecha: 0 2
sentencia (17)
    en la izquierda: 3 4 5, en la derecha: 1 2
operacion (18)
    en la izquierda: 6 7 8, en la derecha: 4 9 10 11 12 13 14 15
opunaria (19)
    en la izquierda: 9 10 11, en la derecha: 7
opbinaria (20)
    en la izquierda: 12 13 14 15, en la derecha: 8



Gramática

    0 $accept: instruccion $end

    1 instruccion: sentencia
    2            | instruccion sentencia

    3 sentencia: END
    4          | operacion END
    5          | error END

    6 operacion: CADENA
    7          | opunaria
    8          | opbinaria

    9 opunaria: operacion OP_SHIFTL
   10         | operacion OP_SHIFTR
   11         | LPAREN operacion RPAREN
```

```
12 opbinaria: operacion OP_CONCAT operacion
13          | operacion OP_NCONCAT NUMERO
14          | operacion OP_SHIFTNL NUMERO
15          | operacion OP_SHIFTNR NUMERO
```

Acciones Semánticas:

Se utilizaron las siguientes funciones para manipular la entrada de texto devuelta por el contenido semántico de cada token, éstas fueron utilizadas a lo largo de cada operación binaria y unarias en la gramática:

```
char* repeat(char* cadena,int n); /* Copia y concatena una cadena n veces */

char* shiftnleft(char* cadena,int n); /*Copia n caracteres del final de la cadena
y las coloca al inicio*/

char* shiftnright(char* cadena,int n); /*Copia n caracteres del inicio de la
cadena y las coloca al final*/
```

Tabla de Símbolos:

Debido a que el análisis sintáctico no involucraba muchos tipos de datos a la vez, además de no incluir entornos y demás, se optó por utilizar una estructura UNION que contuviera los dos tipos de datos atómicos utilizados; la cadena de caracteres y el entero simple. A continuación la definición:

```
%union {
  int intValue; // Valor que representa el entero leído por yylex
  char* stringValue; // Valor que representa la cadena leída por yylex
}
```

Compilación:

Para GNU/Linux seguir instrucciones del archivo INSTALL contenido en el tarball de distribución. por lo general no se necesita nada más que:

```
$ ./configure
$ make -k Makefile
```