Universidad de San Carlos de Guatemala
Facultad de Ingeniería

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

Segundo Proyecto: Documentación

Erik Vladimir Girón Márquez
Carnet # 200313492
Sección C

Guatemala, 2 de Mayo de 2006

Introducción:

Godzilla es interprete para un lenguaje incrustado dentro de un documento HTML(muy al estilo ASP o PHP), que permite la generación de código dentro del mismo archivo HTML mediante instrucción de impresión; incluyendo además un entorno gráfico de desarrollo.

Esta aplicación permite la declaración de variables de 3 tipos diferentes, asignación de éstas, así como la interpretación de construcciones básica de los lenguajes tipo ALGOL (como por ejemplo sentencias *if-else, while y for*), además de permitir evaluar expresiones aritméticas y booleanas.

En este documento se incluye un manual de usuario en las primeras 4 páginas para que sirva de guía paso a paso al usuario a través de los diferentes comandos del programa, sin embargo la interfaz es tan simple que permite un fácil manejo de los comandos del programa, ya sea desde el ratón, o utilizando teclas de acceso rápido.

Se incluye además una referencia técnica de la página 5 en adelante, que intenta explicar de una manera objetiva el funcionamiento interno del programa, describiendo las estructuras, clases y funciones utilizadas en éste, además de describir las librerías necesarias para la compilación del mismo.

El programa fue desarrollado en lenguaje C (para el analizador) y C++ (para el GUI), compilado bajo GNU/GCC 4 y 3.5 para las plataformas GNU/LINUX y MS-Windows respectivamente y distribuido bajo la licencia GPL.

MANUAL DE USUARIO:

Requisitos del Sistema:

- Ordenador x86 o compatible, con tarjeta gráfica, Unidad de CD-ROM y Ratón Funcionales.
- 64 MB de RAM bajo GNU/Linux y 128 MB de memoria RAM bajo MS-Windows.
- Sistema operativo Linux(deseable) o Windows 2000/XP/Vista (NOTA: AUNQUE ESTA DESARROLLADO PARA SOPORTARLAS, POR SU INESTABILIDAD, NO SE RECOMIENDA EN ABSOLUTO WINDOWS 9x O ME).
- En GNU/LINUX es necesario tener instalado correctamente las librerías de QT v3.3+, *además de algún entorno de escritorio(tip, si usa KDE v3.3+ como entorno de escritorio, quiere decir que ya tiene instalado QT)*.

(NOTA: Asegúrese de cumplir los requisitos de hardware y software, o el programa no correrá como debería. también de tener la versión correcta de QT en GNU/Linux o el programa nunca correrá.)

Instrucciones de Uso:

Pre-Pasos:

- Encienda su computador y verifique que cumple con los requisitos del sistema.
- Ingrese a su sistema operativo y verifique que tiene instalado las libreria QT si usa GNU/Linux

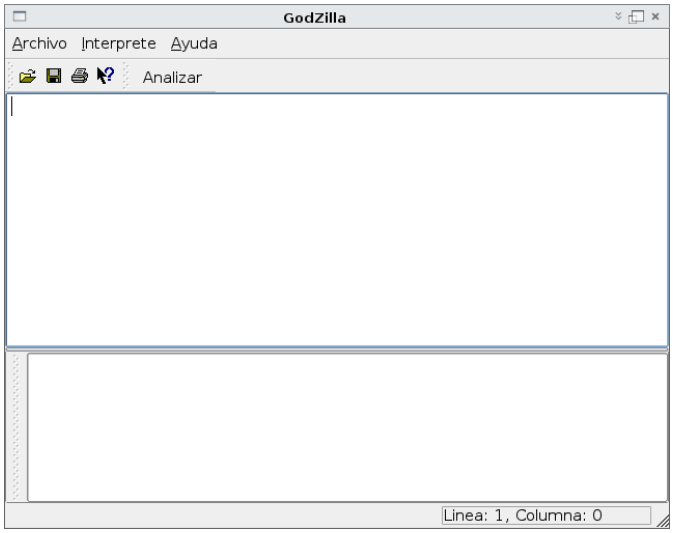


Fig 1

Pasos:

1. Inserte en la unidad de CD-ROM, el disco del programa.
2. Si no se auto ejecuta el programa diríjase a *Unidad de CD/DVD-ROM* e ingrese al directorio bin.
3. Haga doble clic sobre **`godzilla`**(en GNU/LINUX) o sobre **`godzilla.exe`** (en Windows).
4. Inmediatamente se mostrará la ventana principal (figura 1).

- También podra ejecutar desde consola el program con la siguiente sinapsis

  $ ./**`godzilla <file_input> <file_output> <file_error>`**

  donde: file_input: archivo de entrada (en formato ASCII de texto plano)
  file_output: archivo de salida (ASCII)
  file_error: archivo de salida para errores (ASCII)

Ventana del Editor:

Editor:
Aqui es donde se va a escribir el codigo

Browser:
Aqui se mostrara la salida generada
por el interprete al presionar
el boton analizar

Ejemplo de Uso:

- Escriba lo siguiente en el editor de código

```
<html>
<head><title></title></head>
<body>
<%
var i:integer;

for i:=1 to 5
      print('<p> <b>renglon No',i,'</b></p>');
%>
</body>
</html>
```

- Haga clic en el menú Archivo.Guardar Como... e ingrese el nombre con que desea guardar el documento recién creado
- Luego de haber guardado el documento, haga clic en el botón Analizar.
- Se le pedirá nombre para el archivo de salida. ingrese cualquier nombre y con extensión HTML
- Si no hubieron errores en el análisis, se le mostrará el archivo resultante en la ventana de Browser(fig 2).
- Si hubieron errores en el análisis, se le mostrarán los errores en la ventana de browser(fig 3).

Fig 2

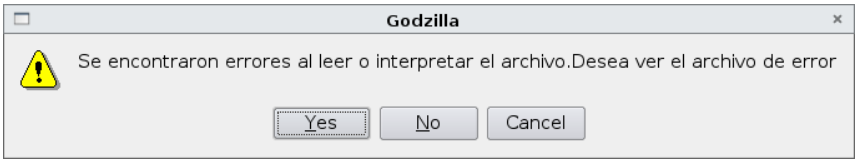


Fig 3

MANUAL TÉCNICO:

Solución Lógica:

En esta sección se hace referencia a las técnicas utilizadas durante el desarrollo del programa, presentando descripciones generales de las estructuras de datos bases y algoritmos utilizados dentro del código fuente. Si desea ver mas información, podrá ver la documentación generada por Doxygen dentro del directorio doc en el disco de distribución, o bien podrá revisar el código fuente.

Lista de archivos

Lista de todos los archivos con descripciones breves:

ast.c [código]	Implementacion del arbol de sintaxis abstracta
ast.h [código]	Definiciones y estructura del arbol de sintaxis abstracta
browser.cpp [código]	Implementacion de la clase browserDock
browser.h [código]	Definiciones de la clase browserDock
colaerr.c [código]	Implementacion de la cola almacenadora de errores
colaerr.h [código]	Definiciones de la cola almacenadora de errores
constantes.h [código]	Constantes utilizadas por el arbol de sintaxis abstracta
godzilla.cpp [código]	Implementacion de la Widged principal del GUI
godzilla.h [código]	Defiinicion de la Widged principal del GUI
main.cpp [código]	Punto de entrada del programa
parserheader.h [código]	Interfaz entre el GUI y el parser
symtab.c [código]	Implementacion de la tabla de simbolos.Incluye la implementacion de rutinas de insercion, busqueda y eliminacion
symtab.h [código]	Estructuras de la tabla de simbolos.Incluyendo rutinas de insercion, busqueda y eliminacion

Lista de componentes

Lista de las clases, estructuras, uniones e interfaces con una breve descripción:

asignacion	Clase de almacenamiento de una asignacion en el AST
ast	Estructura del arbol abstracto de sintaxis (AST), basico para poder evaluar construcciones iterativas del lenguaje
BrowserDock	DockWindow del browser empotrable en la ventana principal, heredando de QDockWindow
colaErr	Cola de errores
constante	Clase de almacenamiento de constantes en el AST
declaracion	Clase de almacenamiento de raiz de una declaracion en el AST

enunciadoFor	Clase de almacenamiento de enunciados for en el AST
enunciadoIf	Clase de almacenamiento de enunciados If en el AST
enunciadoWhile	Clase de almacenamiento de enunciados while en el AST
expr	Clase de almacenamiento de raiz de un arbol de expresiones en el AST
GodZilla	
nodoColaErr	Nodo de la cola de errores
nodoHijo	Nodo a usar para pasar entre producciones de la sintaxis, y paso de expresiones al recorrer un arbol de expresiones
operacion	Clase de almacenamiento de operaciones en el AST
printCall	Clase de almacenamiento de una llamada a Print o a vartablasimbolos en el AST
raiz	Punto de entrada del AST
sentencia	Clase de almacenamiento de raiz de un arbol de sentencias en el AST, siendo a su vez una lista
symbol	Nodo de la tabla de simbolos
symtab	Tabla de simbolos usando modelo de lista simple
tipoError	Clase de almacenamiento de error
token	Clase de almacenamiento de raiz de una lista enlazada de Tokens a imprimir con el comando Print en el AST
variable	Clase de almacenamiento de variables en el AST

Referencia de la Estructura ast

Estructura del arbol abstracto de sintaxis (AST), basico para poder evaluar construcciones iterativas del lenguaje. Más...

```
#include <ast.h>
```

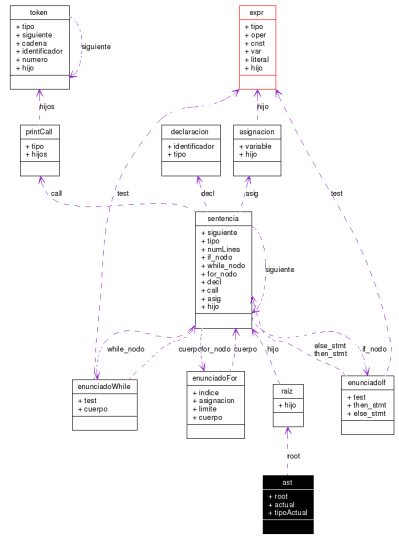


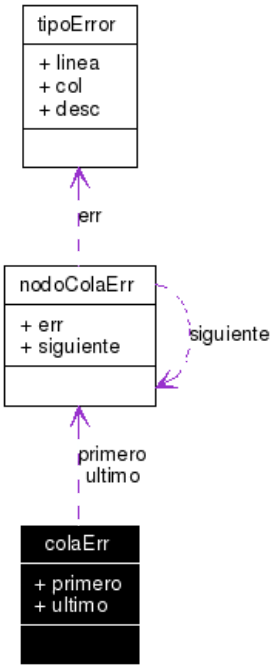
Diagrama de colaboración para ast:

Referencia de la Estructura colaErr

a de errores. Más...

`#include <colaerr.h>`

Diagrama de colaboración para colaErr:

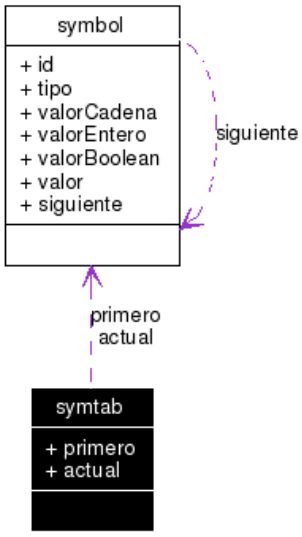


Referencia de la Estructura symtab

a de simbolos usando modelo de lista simple. Más...

`#include <symtab.h>`

Diagrama de colaboración para symtab:



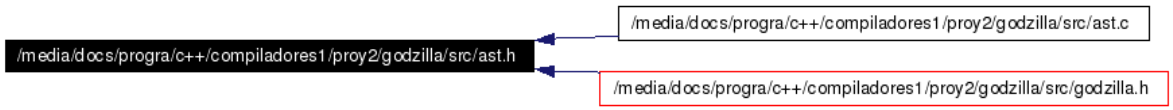
Referencia del Archivo ast.h

Definiciones y estructura del arbol de sintaxis abstracta. Más...

```
#include "constantes.h"
#include "symtab.h"
#include "colaerr.h"
```

Dependencia gráfica adjunta para ast.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

struct variable
Clase de almacenamiento de variables en el AST. Más...

struct constante
Clase de almacenamiento de constantes en el AST. Más...

struct operacion
Clase de almacenamiento de operaciones en el AST. Más...

struct enunciadoIf
Clase de almacenamiento de enunciados If en el AST. Más...

struct enunciadoWhile
Clase de almacenamiento de enunciados while en el AST. Más...

struct enunciadoFor
Clase de almacenamiento de enunciados for en el AST. Más...

struct expr
Clase de almacenamiento de raiz de un arbol de expresiones en el AST. Más...

struct asignacion
Clase de almacenamiento de una asignacion en el AST. Más...

struct printCall
Clase de almacenamiento de una llamada a Print o a vartablasimbolos en el AST. Más...

struct sentencia
Clase de almacenamiento de raiz de un arbol de sentencias en el AST, siendo a su vez una lista. Más...

struct token
Clase de almacenamiento de raiz de una lista enlazada de Tokens a imprimir con el comando Print en el AST. Más...

struct declaracion
Clase de almacenamiento de raiz de una declaracion en el AST. Más...

struct nodoHijo
Nodo a usar para pasar entre producciones de la sintaxis, y paso de expresiones al recorrer un arbol de expresiones. Más...

struct raiz
punto de entrada del AST Más...

struct ast
Estructura del arbol abstracto de sintaxis (AST), basico para poder evaluar construcciones iterativas del lenguaje. Más...

Definiciones

#define yyout embedout
Aliases de yacc en QT, cambiar a nombre de parser a utilizar.

Tipos definidos

typedef sentencia sentencia
typedef variable variable
typedef constante constante
typedef operacion operacion
typedef enunciadoIf enunciadoIf
typedef enunciadoWhile enunciadoWhile
typedef enunciadoFor enunciadoFor
typedef expr expr
typedef asignacion asignacion
typedef printCall printCall
typedef token token
typedef declaracion declaracion
typedef enunciado enunciado
typedef nodoHijo nodo
typedef raiz raiz
typedef ast ast

Funciones

nodo * insertarConstante (int, int)
Agrega una constante al arbol de expresiones.

nodo * insertarVariable (char *)
Agrega una variable al arbol de expresiones.

nodo * insertarCadena (char *)
Agrega una cadena al arbol de expresiones.

nodo * insertarOperacion (int, nodo *, nodo *)
Inserta una operacion al arbol de expresiones.

nodo * insertarExpresion (nodo *)
Agrega un nuevo arbol de expresiones al arbol principal.

nodo * insertarAsignacion (char *, nodo *)
Agrega una asignacion al arbol principal.

nodo * insertarSentencia (int, nodo *, int)
agrega una sentencia al arbol principal

nodo * insertarDeclaracion (char *, int)
agrega una nuva declaracion al arbol principal

nodo * insertarEnunciadoIf (nodo *, nodo *, nodo *)
Agrega un nuevo enunciado if al arbol de sentencias.

nodo * insertarCicloWhile (nodo *, nodo *)
Agrega un nuevo ciclo while ar arbol de sentecias.

nodo * insertarCicloFor (char *, int, int, nodo *)
Agrega un nuevo ciclo for al arbol de sentencias.

nodo * insertarToken (int, void *)
Agrega un nuevo token al arbol de sentencias.

nodo * insertarLlamada (nodo *)
Agrega un nueva llamada de impresion a yyout al arbol de sentencias.

nodo * insertarLlamadaSymTab (int)
Agrega un nueva llamada de impresion de tabla de simbolos al arbol de sentencias.

nodo * concatenarTokens (nodo *, nodo *)
Concatena tokens para parametros de llamadas.

nodo * concatenarSentencia (nodo *, nodo *)
Concatena sentencias para caminar por estas.

void crearRaiz (nodo *, ast *)
Genera el nodo raiz para el arbol dado en el segundo parametro como punto de inicio del arbol y punto de meta del recorrido sintactico.

nodo * refNodoHijo (int tipo, void *dato)
Crea un nodo que refiere al nodo recien creado para sea utilizado por el nodo anterior a este.

int recorrerArbol (ast *tree, char *filename)
Recorre el arbol y escribe resultado en archivo salida.

int recorrerSentencia (sentencia *s)
Recorre sentencias recursivamente y devuelve resultado acarreado.

int evaluarSentencia (sentencia *s)

Evalua sentencia y selecciona tipo de sentencia a evaluar.

int evaluarDeclaracion (declaracion *d)

Evalua declaracion.

int evaluarAsignacion (asignacion *a)

Evalua asignacion.

nodo * evaluarExpresion (expr *e)

Evalua expresion y selecciona tipo de expresion a evaluar.

nodo * evaluarOperacion (operacion *o)

Selecciona operacion binaria a evaluar.

nodo * evaluarOr (nodo *n1, nodo *n2)

Evalua operacion logica OR.

nodo * evaluarAnd (nodo *n1, nodo *n2)

Evalua operacion logica AND.

nodo * evaluarGT (nodo *n1, nodo *n2)

Evalua operacion comparativa Mayor Que.

nodo * evaluarGET (nodo *n1, nodo *n2)

Evalua operacion comparativa Mayor o igual Que.

nodo * evaluarLT (nodo *n1, nodo *n2)

Evalua operacion comparativa Menor Que.

nodo * evaluarLET (nodo *n1, nodo *n2)

Evalua operacion comparativa Menor o Igual Que.

nodo * evaluarEQ (nodo *n1, nodo *n2)

Evalua operacion comparativa Igual.

nodo * evaluarNEQ (nodo *n1, nodo *n2)

Evalua operacion comparativa Desigual.

nodo * evaluarSuma (nodo *n1, nodo *n2)

Evalua operacion aritmetica Suma, y la concatenacion de cadenas.

nodo * evaluarResta (nodo *n1, nodo *n2)

Evalua operacion aritmetica Resta.

nodo * evaluarMult (nodo *n1, nodo *n2)

Evalua operacion aritmetica Multiplicacion.

nodo * evaluarDiv (nodo *n1, nodo *n2)

Evalua operacion aritmetica Division.

int evaluarIf (enunciadoIf *eif)

Evalua bifurcacion If.

int evaluarWhile (enunciadoWhile *ew)

Evalua bucle While.

int evaluarFor (enunciadoFor *ef)

Evalua bucle For.

int evaluarPrintCall (printCall *pc)

Evalua llamada a imprimir en archivo.

int imprimirTokens (token *t)

Evalua recursivamente lista de tokens a imprimir.

void error (char *err, int tipo, void *dato)

Escribe error semantico hacia cola de errores semanticos.

void borrarArbol (ast *tree)

Funciones de eliminacion logica.

void borrarSentencias (sentencia *s)
Borra nodo sentencia.

void borrarDeclaracion (declaracion *d)
Borra nodo declaracion.

void borrarAsignacion (asignacion *a)
Borrar Asignacion.

void borrarExpresion (expr *e)
Elimina de memoria arbol de expresiones.

void borrarOperacion (operacion *o)
borra nodo operacion de mem

void borrarIf (enunciadoIf *eif)
borra nodo if

void borrarWhile (enunciadoWhile *ew)
borra nodo While

void borrarFor (enunciadoFor *ef)
borra nodo for

void borrarPrintCall (printCall *pc)
borra nodo de llamada a imprimir en archivo

void borrarTokens (token *t)
borra recursivamente listado de tokens

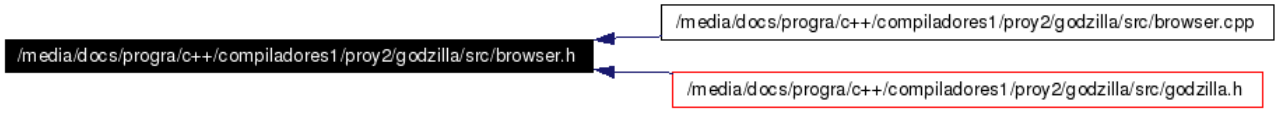
Referencia del Archivo browser.h

Definiciones de la clase browserDock. Más...

```
#include <qdockwindow.h>
#include <qtextbrowser.h>
```

Dependencia gráfica adjunta para browser.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

class BrowserDock

DockWindow del browser empotrable en la ventana principal, heredando de QDockWindow. Más...

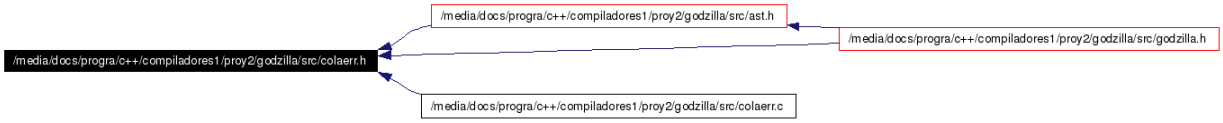
Referencia del Archivo colaerr.h

Definiciones de la cola almacenadora de errores . Más...

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

Dependencia gráfica adjunta para colaerr.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

struct tipoError
Clase de almacenamiento de error. Más...

struct nodoColaErr
Nodo de la cola de errores. Más...

struct colaErr
Cola de errores. Más...

Tipos definidos

typedef nodoColaErr nodoColaErr

typedef colaErr colaErr

typedef tipoError tipoError

Funciones

void encolarError (colaErr *cerr, tipoError *err)
Agrega un error a la cola de errores.

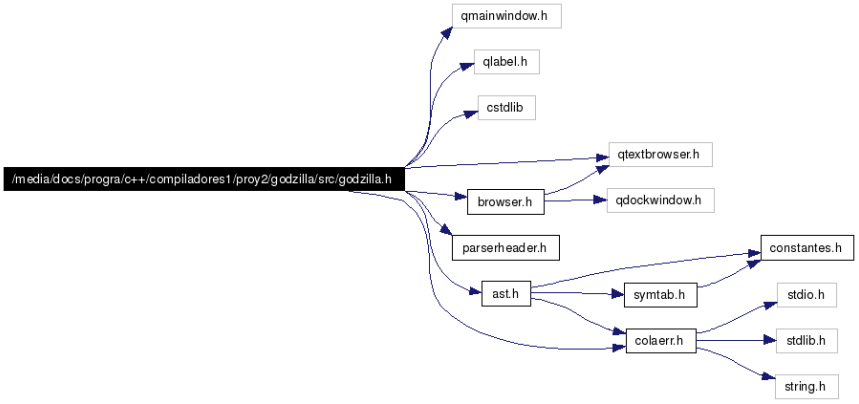
tipoError * sacarError (colaErr *cerr)
Agrega un error a la cola de errores.
void errorLexico (char *msg)
Devuelve primer error en la cola de erroes.
void errorSintactico (char *msg)
Agrega mensaje a la cola de errores lexicos.
void errorSemantico (char *msg, int numlinea)
Agrega mensaje a la cola de errores sintacticos.
int escribirErrorLogXML (const char *filename)
Agrega mensaje a la cola de errores semanticos.

Referencia del Archivo godzilla.h

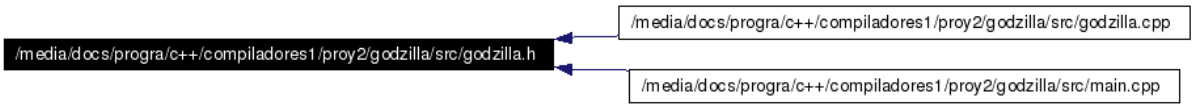
defiinicion de la Widged principal del GUI. Más...

```
#include <qmainwindow.h>
#include <qlabel.h>
#include <cstdlib>
#include <qtextbrowser.h>
#include "browser.h"
#include "parserheader.h"
#include "ast.h"
#include "colaerr.h"
```

Dependencia gráfica adjunta para godzilla.h:



Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Referencia del Archivo parserheader.h

interfaz entre el GUI y el parser. Más...

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:

Ir al código fuente de este archivo.

Funciones

int inputparse (const char *filein, const char *fileout)
    Interfaz para parser y lexer.
int generarSalidaError (const char *fileerr)
    Interfaz para colaerr.

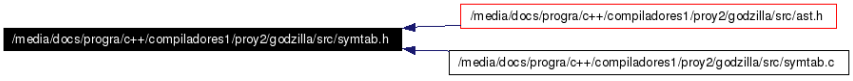
Referencia del Archivo symtab.h

Estructuras de la tabla de simbolos.Incluyendo rutinas de insercion, busqueda y eliminacion. Más...

```
#include "constantes.h"
```

Dependencia gráfica adjunta para symtab.h:

Este gráfico muestra que archivos directa o indirectamente incluyen a este archivo:



Clases

struct symbol
    Nodo de la tabla de simbolos. Más...
struct symtab
    Tabla de simbolos usando modelo de lista simple. Más...

Tipos definidos

typedef symtab symtab
    ESTRUCTURA.
typedef
symbol symbol

Funciones

symbol * insertarSimbolo (char *id, int tipo)
    Inserta un nuevo simbolo sin comprobar su previa existencia.
symbol * buscarSimbolo (char *id)

Busca y devuelve simbolo segun su identificador dado en el param.

void borrarTabla (void)

Elimina todos los registros de la tabla de simbolos.

int openSymtabFile (const char *filename)

Abre archivo hacia donde se imprimira archivo de tabla de simbolos.

int closeSymtabFile (void)

Cierra archivo hacia donde se escribio tabla de simbolos.

int printSymtabFile (int linea)

Imprime tabla de simbolos a archivo dado.

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.

```
    0 $accept: inicio $end

    1 inicio: embed
    2       | inicio embed

    3 embed: OPEN_EMBED codigo CLOSE_EMBED
    4      | error CLOSE_EMBED
    5      | OPEN_EMBED CLOSE_EMBED

    6 codigo: sentencias

    7 sentencias: sentencia
    8           | sentencias sentencia

    9 sentencia: def_var
   10          | asignacion
   11          | condicionales
   12          | ciclos
   13          | llamadas
   14          | error

   15 def_var: KW_VAR TK_IDENTIFICADOR TK_DOSPUNTOS KW_INTEGER TK_PUNTOCOMA
   16        | KW_VAR TK_IDENTIFICADOR TK_DOSPUNTOS KW_BOOLEAN TK_PUNTOCOMA
   17        | KW_VAR TK_IDENTIFICADOR TK_DOSPUNTOS KW_STRING TK_PUNTOCOMA

   18 asignacion: TK_IDENTIFICADOR TK_OP_ASIGNACION expr TK_PUNTOCOMA

   19 condicionales: KW_IF expr KW_THEN bloque KW_ELSE bloque

   20 ciclos: ciclo_while
   21       | ciclo_for

   22 ciclo_while: KW_WHILE expr bloque

   23 ciclo_for: KW_FOR TK_IDENTIFICADOR TK_OP_ASIGNACION TK_NUMERO KW_TO TK_NUMERO
bloque

   24 llamadas: KW_PRINT TK_APAREN lista_tokens TK_CPAREN TK_PUNTOCOMA
   25         | KW_PRINTSYMTAB TK_APAREN TK_CPAREN TK_PUNTOCOMA

   26 lista_tokens: lista_tokens TK_COMA token
   27             | token

   28 token: TK_IDENTIFICADOR
   29      | TK_CADENA
   30      | TK_NUMERO

   31 bloque: KW_BEGIN sentencias bloque KW_END TK_PUNTOCOMA
   32       | KW_BEGIN error KW_END TK_PUNTOCOMA
```

```
33       | sentencia

34 expr: expr_and
35     | expr KW_OP_OR expr_and

36 expr_and: expr_igual
37         | expr_and KW_OP_AND expr_igual

38 expr_igual: expr_relacional
39           | expr_igual TK_OP_EQ expr_relacional
40           | expr_igual TK_OP_NOEQ expr_relacional

41 expr_relacional: expr_suma
42                | expr_relacional TK_OP_GT expr_suma
43                | expr_relacional TK_OP_GET expr_suma
44                | expr_relacional TK_OP_LT expr_suma
45                | expr_relacional TK_OP_LET expr_suma

46 expr_suma: expr_mult
47          | expr_suma TK_OP_SUMA expr_mult
48          | expr_suma TK_OP_RESTA expr_mult

49 expr_mult: term
50          | expr_mult TK_OP_MULT term
51          | expr_mult TK_OP_DIV term

52 term: TK_NUMERO
53     | KW_TRUE
54     | KW_FALSE
55     | TK_APAREN expr TK_CPAREN
56     | TK_APAREN error TK_CPAREN
57     | TK_IDENTIFICADOR
58     | TK_CADENA
```

Descripcion de las librerías utilizadas:

Para el análisis léxico/sintáctico/semántico se utilizó únicamente las librerías estándares de C (stdio, stdlib, string, etc).

Para el desarrollo de la interfaz gráfica se requirió de una librería que permitiera el desarrollo rápido de éste, además que permitiera la potabilidad del código entre diferentes plataformas. Para ésto se tenía 3 opciones:
a) Java,
b)GTK+,
c)QT.
Dado que la opción de Java no era factible, ya que implicaría llamadas a código en C, lo cual requería de re implementar el código para hacerlo compatible con el JNI, lo cual retrasaría el tiempo de desarrollo.

Teníamos como segunda opción GTK+, la cual hubiese podido ser tomada por su alta eficiencia y sin hacer casi ningún cambio al código. Sin embargo no fue tomada, ya que al momento de desarrollo no tenía instaladas las librerías de desarrollo en GTK.

Se optó entonces por las librerías **QT**, las cuales, aunque implicaría cambiar levemente el código en C para hacerlo compatible con el ISO C++, permitió de una manera sencilla, rápida y amigable el desarrollo de la aplicación, además de proveer una de las características que más interesaban, la portabilidad entre diferentes plataformas.

Compilacion

Requisitos: GCC v 3.5+ y QT v 3.3+ correctamente instalados, ademas de las utilidades de desarrollo necesarias (make, etc).

Para compilar desde GNU/Linux , ingresar desde la consola hacia el directorio del código fuente, y hacer lo siguiente:

```
$ qmake -o Makefile godzilla.pro
$ qmake -o /src/Makefile /src/src.pro
$ make clean
$ make
```

Conclusiones:

Herramientas como Yacc y Lex permitieron que el desarrollo del analizador fuese más rápido, enfocando parte del tiempo en la creacion de una representación intermedia adecuada, y no en la implementación de una gramática hecha en C, por lo que se logró implementar de manera correcta un árbol de sintaxis abstracta como representación intermedia del análisis semántico, lo que permitió gran flexibilidad a la hora de evaluar enunciados iterativos, que de otra manera serían imposibles de analizar en un parser de una pasada y libre del contexto.

Debido a que las librerias actuales para el desarrollo de entornos gráficos son muy sencillas de utilizar, permitió un desarrollo enfocado más al funcionamiento interno de la aplicacion.