MANUAL TÉCNICO
“Interprete de Matrices”

Compiladores

Tecla Parra Roberto

MANUAL TÉCNICO “PROYECTO FINAL”

ALUMNOS:
Aguirre Cruz Eder Jonathan
Saulés Cortés Jhonatan

GRUPO: 3CM1

CONTENIDO

INTRODUCCIÓN

La finalidad de todo manual técnico es la de proporcionar al lector la lógica con la que se ha desarrollado una aplicación, la cual se sabe que es propia de cada programador; por lo que se considera necesario ser documentada

Este manual describe los pasos necesarios para cualquier persona que tenga ciertas bases de sistemas pueda realizar la instalación de la aplicación, creada para establecer la comunicación entre dos personas a través de un canal de comunicación seguro. El presente manual técnico tiene como finalidad describir el diseño del prototipo para la gestión del interprete para matrices en el entorno yacc.

La implementación del interprete para matrices en el entorno yacc. se basa en una adaptación para el uso de operaciones de matrices matemáticas, es decir un pequeño interprete que realiza operaciones básicas de un compilador, y éstas han dado excelentes resultados de uso. Las aplicaciones de éste tipo, ayudan a comprender el funcionamiento de un compilador al utilizar la sintaxis que un compilador conlleva.

La implementación de la aplicación cuenta con una pequeña simulación de lo que es el funcionamiento de un compilador. Resulta ser bastante fácil de implementar puesto solo es necesario instalar los paquetes básicos disponibles en cualquier distribución de Ubuntu.
Es importante tener en cuenta que en el presente manual se hace mención a las especificaciones mínimas de hardware y software para la correcta instalación de la aplicación.

OBJETIVO:

Proporcionar una guía para el lector, del desarrollo de la interfaz y de la instalación del interprete para matrices en el entorno yacc.

Aspectos del Análisis

La implementación del interprete para matrices en el entorno yacc. Se basa en una adaptación para el uso de operaciones de matrices matemáticas, es decir un pequeño interprete que realiza operaciones básicas de un compilador, y éstas han dado excelentes resultados de uso. Las aplicaciones de éste tipo, ayudan a comprender el funcionamiento de un compilador al utilizar la sintaxis que un compilador conlleva.

La aplicación web se puede utilizar en dispositivos PC que soporten el sistema operativo LINUX en cualquiera de sus versiones. En estos tiempos en que la tecnología ha avanzado la mayoría de los dispositivos pueden accesar a la obtención de dicho sistema operativo.
La aplicación está orientada a éste sistema operativo debido a que es más confiable y es mayormente utilizado para el desarrollo de aplicaciones del mismo tipo que la nuestra. Es por ello que esta aplicación va orientada a ese tipo de dispositivos. Una de las ventajas que se presentan en este tipo de tecnología es que tiene mayor seguridad, confiabilidad y por eso muchos usuarios la utilizan.

El motivo de usar este sistema operativo, se debe a la creciente popularidad la cual incrementa a un ritmo exponencial que conlleva esta plataforma respecto a sus competidores, también otra de las grandes ventajas que tiene esta plataforma es que se habla de un software libre

Especificaciones Técnicas

1. REQUERIMIENTOS MÍNIMOS DE HARDWARE

- Procesador: Core
- Memoria RAM: 1 Gigabytes (GB) (Mínimo)
- Disco Duro: 500 Gb.

2. REQUERIMIENTOS MÍNIMOS DE SOFTWARE

- Sistema Operativo: Linux Ubuntu 12.04 Precise Pangolin o versiones superiores.
- Lenguaje de Programación: Java, Yacc
- Entorno de programación y compilador: GNU/java byacc, bison

Herramientas utilizadas para el desarrollo

- **Lenguaje JAVA**

  El lenguaje para la programación en Java, es un lenguaje orientado a objeto, de una plataforma independiente. Java es un lenguaje de programación y una plataforma informática comercializada por primera vez en 1995 por Sun Microsystems. Hay muchas aplicaciones y sitios web que no funcionarán a menos que tenga Java instalado y cada día se crean más. Java es rápido, seguro y fiable. Desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet, Java está en todas partes.

  Esta programación Java tiene muchas similitudes con el lenguaje C y C++, asi que si se tiene conocimiento de este lenguaje, el aprendizaje de la programación Java será de fácil comprensión por un programador que haya realizado programas en estos lenguajes.

- **YACC**

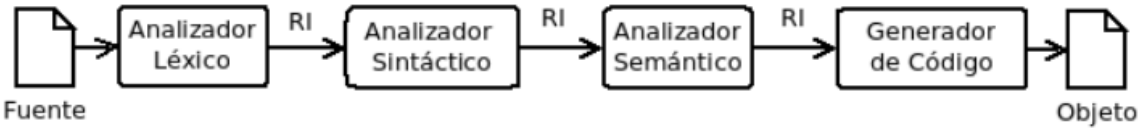
  Yacc es un programa para generar analizadores sintácticos. Las siglas del nombre significan Yet Another Compiler-Compiler, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica escrita en una notación similar a la BNF. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C.

Fue desarrollado por Stephen C. Johnson en AT&T para el sistema operativo Unix. Después se escribieron programas compatibles, por ejemplo Berkeley Yacc, GNU bison, MKS yacc y Abraxas yacc (una versión actualizada de la versión original de AT&T que también es software libre como parte del proyecto de OpenSolaris de Sun). Cada una ofrece mejoras leves y características adicionales sobre el Yacc original, pero el concepto ha seguido siendo igual. Yacc también se ha reescrito para otros lenguajes, incluyendo Ratfor, EFL, ML, Ada, Java, y Limbo.

Puesto que el analizador sintáctico generado por Yacc requiere un analizador léxico, se utiliza a menudo conjuntamente con un generador de analizador léxico, en la mayoría de los casos lex o Flex, alternativa del software libre. El estándar de IEEE POSIX P1003.2 define la funcionalidad y los requisitos a Lex y Yacc.

- ## Arquitectura del compilador

La arquitectura clásica de un compilador está compuesta por cuatro fases las cuales transforman gradualmente un fichero fuente de entrada en un fichero objeto de salida. Estas fases son realizadas por cuatro módulos independientes denominados Analizador Léxico, Analizador Sintáctico, Analizador Semántico y Generador de Código. Los módulos se comunican entre sí mediante algún tipo de representación intermedia del programa fuente.



INSTALACIÓN DE YACC EN UBUNTU/LINUX

- **Instalación de lex / flex.**

Instalación en Linux.

Lex está instalado en los ordenadores con este sistema operativo que el alumno encontrará en el laboratorio.

Si el alumno dispone de un equipo propio en el que tenga instalado Linux, es probable que ya disponga de este paquete y, en caso contrario, seguro que conoce la manera de instalarlo ya que esta incluido en las distribuciones de Linux estándares.

Primero instalaremos Flex, para hacerlo abrimos una terminal y escribimos:

~> sudo zypper in flex

Les pedirá el password del usuario root, la ingresan e iniciara la instalación.

Par instalar Bison el proceso es el mismo escribimos en la terminal

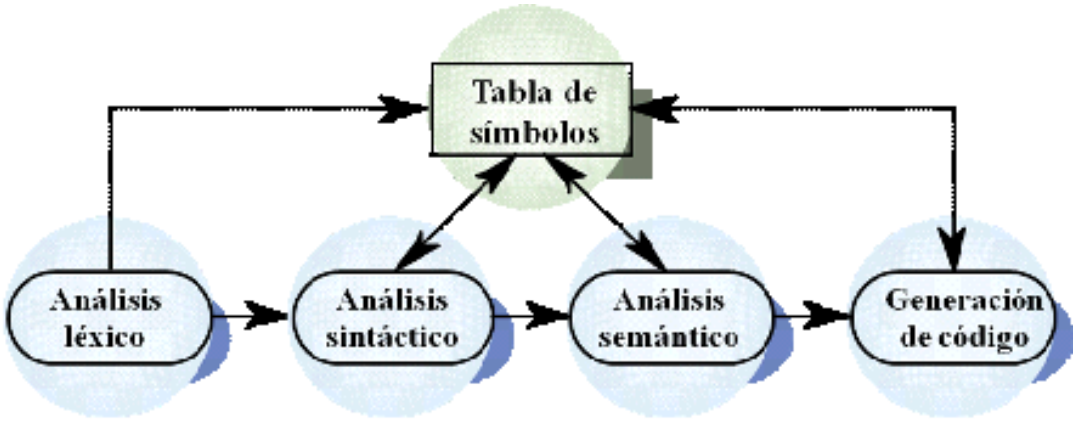
~> sudo zypper in bison

Y listos, flex y bison instalados.

Para los que utilicen Ubuntu o una distribución Debian usan los comandos

sudo apt-get install flex
sudo apt-get install bison

ARQUITECTURA PRESENTADA PARA LA APLICACIÓN



ESPECIFÍCACION DE LOS ARCHIVOS DE LA APLICACIÓN (Proyecto.y, TablaDeSimbolos.java y MaquinaDePila.java)

Proyecto.y	
%token IF %token ELSE %token WHILE %token FOR %token COMP %token DIFERENTES %token MAY %token MEN %token MAYI %token MENI %token FNCT %token NUMBER %token VAR	/* Seccion para declarar los token's utilizados en las gramáticas del intérprete de matrices, dichos token's estarán presentes en todas las operaciones que dicho interprete pueda realizar*/

```
%token AND

%%

list:
    | list'\n'
| list linea '\n'
;

linea: exp ';' {$$ = $1;}
    |linea exp ';' {$$ = $1;}
;

exp:  VAR {
$$ = new ParserVal(maquina.agregarOperacion("varPush_Eval"));
maquina.agregar($1.sval);
}
|NUMBER {
$$ = new ParserVal(maquina.agregarOperacion("constPush"));
maquina.agregar($1.dval);
}
| VAR '[' initNum ']' '=' '{' listaDeListas '}' {
matrizAux = new Matriz(auxiliar);
$$ = new ParserVal(maquina.agregarOperacion("constPush"));
maquina.agregar(matrizAux);

maquina.agregarOperacion("varPush");
maquina.agregar($1.sval);
maquina.agregarOperacion("asignar");

maquina.agregarOperacion("varPush_Eval");
maquina.agregar($1.sval);
}
| VAR '=' exp {
$$ = new ParserVal($3.ival);

maquina.agregarOperacion("varPush");
maquina.agregar($1.sval);
maquina.agregarOperacion("asignar");

maquina.agregarOperacion("varPush_Eval");
maquina.agregar($1.sval);
}
```

/* Seccion donde declaramos las gramáticas correspondientes al interprete, las cuáles hacen posible la interpretación de la sintaxis introducida a través de línea de comando de Ubuntu, estas operaciones leídas e interpretadas por las gramáticas, se agregaran a la máquina de pila, donde se tienen guardas las acciones correspondientes que van a ser introducidas a la pila de yacc */

```
                    | exp '*' exp {
                    $$ = new ParserVal($1.ival);

                maquina.agregarOperacion("multiplicar");
                                    }
                    | exp '+' exp {
                    $$ = new ParserVal($1.ival);

                maquina.agregarOperacion("sumar");
                                    }
                    | exp '-' exp {
                    $$ = new ParserVal($1.ival);

                maquina.agregarOperacion("restar");
                                    }
```

TablaDeSimbolos.java

```
public Object encontrar(String nombre){
    for(int i = 0; i < simbolos.size(); i++)
    if(nombre.equals(simbolos.get(i).getNombre()))
        return simbolos.get(i).getObjeto();
    return null;
    }

public boolean insertar(String nombre, Object objeto){
    Par par = new Par(nombre, objeto);
    for(int i = 0; i < simbolos.size(); i++)
    if(nombre.equals(simbolos.get(i).getNombre())){
        simbolos.get(i).setObjeto(objeto);
        return true;
        }
    simbolos.add(par);
    return false;
    }

    public void Mostrar(){
    for(int i = 0; i < simbolos.size(); i++){
    System.out.println(simbolos.get(i).getNombre() +
    simbolos.get(i).getObjeto().toString());
    }
```

/* En la tabla de símbolos, se agregaran las funciones de insertar, mostrar y encontrar, que son parte fundamental para el funcionamiento de las gramáticas, estas funciones están programadas por defecto para que se mantengan siempre a disposición de la línea de comando */