

PROCESADORES DE LENGUAJES

Bison y flex

Prof. Dr. Nicolás Luis Fernández García
Departamento de Informática y Análisis Numérico

Grado de Ingeniería Informática
Especialidad de Computación
Tercer curso. Segundo cuatrimestre
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

1 Reconocimiento de expresiones aritméticas simples

- Descripción
- Funcionamiento del intérprete
- Ficheros y subdirectorios
- Autómata finito determinista que reconoce prefijos viables

Reconocimiento de expresiones aritméticas simples

Descripción

Descripción (1/2)

- **Expresiones aritméticas** compuestas solamente por **números** y que terminan con un salto de línea.
- Operaciones permitidas:
 - **Suma:** $2 + 3$
 - **Resta:** $2 - 3$
 - **Multiplicación:** $2 * 3$
 - **División:** $2 / 3$
 - **Paréntesis y combinación de operadores:** $4 * (3 + 2) / (5 - 1)$

Reconocimiento de expresiones aritméticas simples

Descripción

Descripción (2/2)

- Comprueba si las expresiones aritméticas son **léxica** y **sintácticamente** correctas.
- Muestra un **mensaje** cuando se detecta un **error**

Se usa el comando **%error-verbose** en el fichero **interpreter.y** para mostrar más información de un **error**.

Reconocimiento de expresiones aritméticas simples

Descripción

Observaciones

- **No** permite operadores **unarios**: se permitirá en el ejemplo n° 3.
 - Signo “+” unario: + 2
 - Signo “-” unario: - 2
- **No evalúa** las expresiones: se hará en el ejemplo n° 4.

Contenido del ejemplo

1 Reconocimiento de expresiones aritméticas simples

- Descripción
- **Funcionamiento del intérprete**
- Ficheros y subdirectorios
- Autómata finito determinista que reconoce prefijos viables

Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Compilación

- **make**
 - Se genera el fichero **interpreter.exe**

Compilación

```
$ make
Accessing directory parser

make[1]: se entra en el directorio ...
Generating:  interpreter.tab.c interpreter.tab.h

Compiling:  interpreter.tab.c
Generating:  lex.yy.c
Compiling:  lex.yy.c
...
Compiling  interpreter.cpp

Generating  interpreter.exe
```


Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Ejecución

- Desde la **línea de comandos**.
- **Redirigiendo** un fichero de entrada.

Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Ejecución

- Desde la línea de comandos.

Ejecución desde la línea de comandos

```
$ ./interpreter.exe
stmtlist --> epsilon
2 + 3
exp --> NUMBER
exp --> NUMBER
exp --> exp '+' exp
stmtlist --> stmtlist exp '\n'
Correct expression
```

Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Ejecución

- Fin del intérprete: uso de **Control D**

Ejecución

```
Control D  
>>>>>> End of file <<<<<<<<  
program --> stmtlist
```

Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Ejecución

- Fin del intérprete: uso de `#` al principio de línea.

Ejecución

```
#  
>>>>>> End of file <<<<<<<<  
program --> stmtlist
```

Reconocimiento de expresiones aritméticas simples

Funcionamiento del intérprete

Ejecución

- **Redirigiendo** un fichero de entrada: uso del operador “<”.

Fichero: test.txt

```
2
2 + 3
2 - 3
2 * 3
2 / 3
(4 + 5) / (8 - 2)
```

Ejecución

```
$ ./interpreter.exe < test.txt

stmtlist --> epsilon
exp --> NUMBER
stmtlist --> stmtlist exp '\n'
Correct expression

stmtlist --> stmtlist '\n'
exp --> NUMBER
exp --> NUMBER
exp --> exp '+' exp
stmtlist --> stmtlist exp '\n'
Correct expression
```

Contenido del ejemplo

- 1 Reconocimiento de expresiones aritméticas simples
 - Descripción
 - Funcionamiento del intérprete
 - **Ficheros y subdirectorios**
 - Autómata finito determinista que reconoce prefijos viables

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Ficheros y subdirectorios (1/2)

- **interpreter.cpp**: programa principal.
- **makefile**: fichero para la compilación del intérprete.
- **Doxyfile**: fichero de configuración de doxygen.
- Subdirectorio **parser**
 - **interpreter.y**: fichero de yacc con la **gramática** del analizador sintáctico.
 - **interpreter.l**: fichero de lex con las **expresiones regulares** del analizador léxico.
 - **makefile**: fichero de compilación del subdirectorio **parser**.

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Ficheros y subdirectorios (2/2)

- Subdirectorio **error**
 - **error.hpp**: prototipos de las funciones de **recuperación de error**.
 - **error.cpp**: código de las funciones de **recuperación de error**.
 - **makefile**: fichero de compilación del subdirectorio **error**.
- Subdirectorio **includes**
 - **macros.hpp**: macros para mejorar la visualización por pantalla.
- Subdirectorio **examples**
 - **test.txt**: fichero de ejemplo **sin** errores.
 - **test-error.txt**: fichero de ejemplo **con** errores.

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

interpreter.cpp

- Programa principal.

interpreter.cpp

```
#include "../parser/interpreter.tab.h"
int lineNumber = 1; //!< Line counter

int main()
{
    /* Option -t needed to debug */
    /* 1, on; 0, off */
    yydebug = 0;

    /* Parser function */
    yyparse();

    /* End of program */
    return 0;
}
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

makefile

- Fichero para la compilación del intérprete.

makefile

```
NAME=interpreter
CPP = g++
CFLAGS = -c -g -Wall -ansi -O2
LFLAGS = -lfl

OBJECTS= $(NAME).o
OBJECTS-PARSER = parser/*.o
OBJECTS-ERROR = error/*.o

INCLUDES = ./parser/interpreter.tab.h

all: $(NAME).exe

$(NAME).exe: parser-dir error-dir $(OBJECTS)
    @echo "Generating " $@
    @$$(CPP) $(OBJECTS) $(OBJECTS-PARSER) $(OBJECTS-ERROR) $(LFLAGS) -o $@

...
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio parser: interpreter.l (1/2)

- Definiciones regulares.

interpreter.l

DIGIT [0-9]

NUMBER1 {DIGIT}+\.?

NUMBER2 {DIGIT}*\. {DIGIT}+

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **parser**: interpreter.l (2/2)

- Expresiones regulares.

interpreter.l

```
[ \t]    { ; }    /* skip white space and tabulator */
'\n'    { /* Line counter */ lineNumber++;
          return '\n';
        }

{NUMBER1}|{NUMBER2} { return NUMBER; }

^# { /* End: # at the beginning of the line */
    return 0;
  }

<<EOF>> { /* End */ return 0; }

.       { /* Any other character */
          return yytext[0];
        }
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **parser**: interpreter.y

- Reglas de la gramática

interpreter.y

```
program : stmtlist
        ;

stmtlist: /* Empty: epsilon rule */
        | stmtlist '\n' /* Empty line */
        | stmtlist exp '\n'
        | stmtlist error '\n'
        ;

exp: NUMBER
   | exp '+' exp
   | exp '-' exp
   | exp '*' exp
   | exp '/' exp
   | '(' exp ')'
   ;
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **parser**: makefile

- Fichero de compilación del subdirectorio **parser**.

makefile

```
NAME=interpreter
CPP = g++
CFLAGS = -c -g -Wall -ansi -O2
LEX = flex
YACC= bison
YFLAGS = -d -t -g
...
lex.yy.c: $(NAME).l $(NAME).tab.h $(INCLUDES)
    @echo "Generating: " $@
    @$(LEX) $<
    @echo
...
$(NAME).tab.c $(NAME).tab.h: $(NAME).y $(INCLUDES)
    @echo "Generating: " $(NAME).tab.c $(NAME).tab.h
    @$(YACC) $(YFLAGS) $<
    @echo
...
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **error**: error.hpp

- Prototipos de las funciones de **recuperación de error**.

error.hpp

```
#ifndef _ERROR_HPP_
#define _ERROR_HPP_

#include <string>

void warning(std::string errorMessage1, std::string errorMessage2);

void yyerror(std::string errorMessage);

#endif // _ERROR_HPP_
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **error**: error.cpp

- Código de las funciones de **recuperación de error**.

error.cpp

```
void warning(std::string errorMessage1, std::string errorMessage2)
{
    std::cerr << BIRED;
    std::cerr << " Error line " << lineNumber
                << " --> " << errorMessage1 << std::endl;
    std::cerr << RESET;

    if (errorMessage2.compare("") != 0)
        std::cerr << "\t" << errorMessage2 << std::endl;
}

void yyerror(std::string errorMessage)
{
    warning("Parsing error", errorMessage);
}
```


Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **error**: makefile

- Fichero de compilación del subdirectorio **error**.

makefile

```
NAME=error
CPP = g++
CFLAGS = -c -g -Wall -ansi -O2
OBJECTS= $(NAME).o
INCLUDES = $(NAME).hpp ../includes/macros.hpp

all: $(OBJECTS)
$(NAME).o: $(NAME).cpp $(INCLUDES)
    @echo "Compiling " $<
    @$$(CPP) $$$(CFLAGS) $<
    @echo
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **includes**: macros.hpp

- Macros para mejorar la visualización por pantalla.

makefile

```
#ifndef _MACROS_HPP_
#define _MACROS_HPP_ ///< Macros for the screen
...
#define PLACE(x,y)          printf("\033[%d;%dH",x,y) ///< Place
#define CLEAR_SCREEN      "\33[2J" ///< Clear the screen
#define CLEAR_REST_OF_LINE "\33[K" ///< Clear until the end of line
#define RESET              "\e[0m" ///< Reset
#define INTENSITY         "\e[1m" ///< Intensity
#define FAINT              "\e[2m" ///< Barely perceptible
#define ITALIC             "\e[3m" ///< Italic
#define UNDERLINE         "\e[4m" ///< subrayado
#define BLINK              "\e[5m" ///< Blink
#define INVERSE           "\e[7m" ///< Inverse
...
#endif // _MACROS_HPP_
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **examples**: test.txt

- Fichero de ejemplo **sin** errores.

test.txt

```
2  
  
2 + 3  
2 - 3  
  
2 * 3  
2 / 3  
  
(4 + 5) / (8 - 2)
```

Reconocimiento de expresiones aritméticas simples

Ficheros y subdirectorios

Subdirectorio **examples**: test-error.txt

- Fichero de ejemplo **con errores**.

test-error.txt

```
- 2
+ 2
2 * * 4
4 * 5)
(4 * 5
2 * (5 - 3) / (4 * 5)
```

Contenido del ejemplo

- 1 Reconocimiento de expresiones aritméticas simples
 - Descripción
 - Funcionamiento del intérprete
 - Ficheros y subdirectorios
 - Autómata finito determinista que reconoce prefijos viables

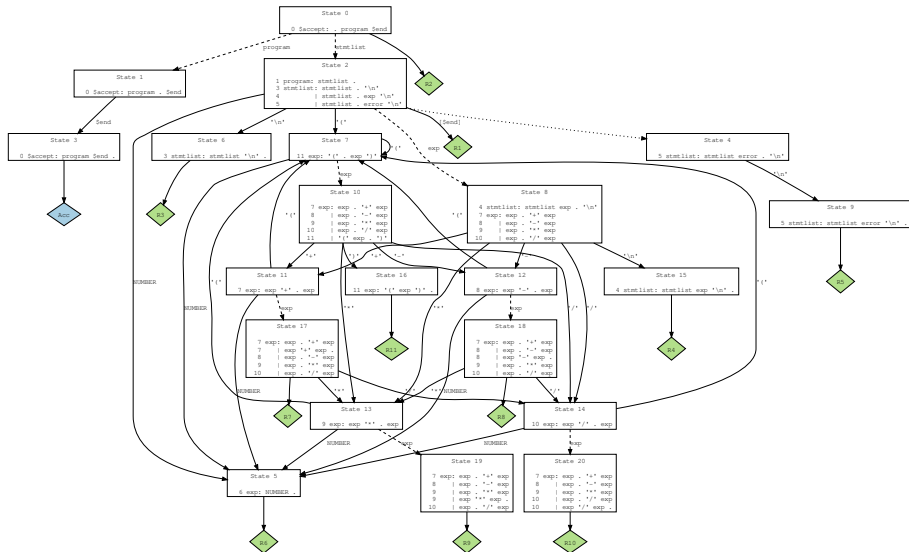
Reconocimiento de expresiones aritméticas simples

Autómata finito determinista que reconoce prefijos viables

Autómata finito determinista que reconoce prefijos viables

- En el directorio **parser**
 - **Representación gráfica**
 - **dot -TXXX interpreter.dot -o interpreter.XXX**
donde XXX puede ser: svg, png, jpg, gif, pdf, ps, fig, ...
 - Ejemplo
dot -Tjpg interpreter.dot -o interpreter.jpg
 - **Fichero de texto y tabla LALR**
 - **bison -v interterper.y**
Se genera el fichero **interpreter.output**

Autómata finito determinista que reconoce prefijos viables



Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 **Análisis de un fichero**
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 2 Análisis de un fichero
 - Novedades
 - Ficheros modificados

Análisis de un fichero

Novidades

Novidades (1/3)

- **Analiza** un fichero de entrada pasado como argumento.
- Muestra el **nombre del programa** en los **mensajes de error**.

Análisis de un fichero

Novedades

Novedades (2/3)

- **Analiza** un fichero de entrada pasado como argumento.

Fichero: test.txt

```
2
2 + 3
2 - 3
2 * 3
2 / 3
(4 + 5) / (8 - 2)
```

Ejecución

```
$ ./interpreter.exe test.txt

stmtlist --> epsilon
exp --> NUMBER
stmtlist --> stmtlist exp '\n'
Correct expression

stmtlist --> stmtlist '\n'
exp --> NUMBER
exp --> NUMBER
exp --> exp '+' exp
stmtlist --> stmtlist exp '\n'
Correct expression
...
```

Análisis de un fichero

Novedades

Novedades (3/3)

- Muestra el **nombre del programa** en los **mensajes de error**.

Ejemplo

```
$ ./interpreter.exe  
afd
```

```
Program: ./interpreter.exe
```

```
Error line 1 --> Parsing error
```

```
syntax error, unexpected $undefined, expecting $end or NUMBER or '\n' or '('
```

Contenido del ejemplo

- 2 Análisis de un fichero
 - Novedades
 - Ficheros modificados

Análisis de un fichero

Ficheros modificados

Ficheros modificados

- **interpreter.cpp**
 - Manejo de **argumentos** desde la línea de comandos
- **error.cpp**
 - Se modifica la función **warning** para que muestre el nombre del programa.

Análisis de un fichero

Ficheros modificados

interpreter.cpp

- Manejo de **argumentos** desde la línea de comandos.

interpreter.cpp

```
int main(int argc, char *argv[])
{
    ...
    if (argc == 2)
        yyin = fopen(argv[1], "r");

    /* Copy the name of the interpreter */
    progame = argv[0];

    /* Parser function */
    yyparse();

    /* End of program */
    return 0;
}
```

Análisis de un fichero

Ficheros modificados

error.cpp

- Se modifica la función `warning` para que muestre el nombre del programa.

error.cpp

```
void warning(std::string errorMessage1,
             std::string errorMessage2)
{
    /* NEW in example 2 */
    std::cerr << IGREEN;
    std::cerr << " Program: " << progame << std::endl;
    ...
}
```


Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios**
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

3 Reconocimiento de operadores unarios

- Novedades
- Ficheros modificados
- Ejercicios

Reconocimiento de operadores unarios

Novedades

Novedades

- Operadores **unarios**.
- **Componentes léxicos o tokens con nombre**.
- Se muestran los **errores léxicos**.

Reconocimiento de operadores unarios

Novedades

Operadores unarios

- Signo “+” unario: + 2
- Signo “-” unario: - 2

Reconocimiento de operadores unarios

Novedades

Operadores unarios: observación

- Permite varios operadores unarios seguidos
++ 2
+ - + 3
- Curiosidad: el lenguaje C también lo permite

Reconocimiento de operadores unarios

Novedades

Componentes léxicos o tokens con nombre

- **NEWLINE**
- **NUMBER**
- **PLUS, MINUS** (asociatividad por la izquierda)
- **MULTIPLICATION, DIVISION** (asociatividad por la izquierda)
- **LPAREN, RPAREN**
- **UNARY** (máxima precedencia)

Reconocimiento de operadores unarios

Novedades

Componentes léxicos o tokens con nombre

- Funcionamiento

test.txt

```
+ 2 * (-3 + 5)
...
```

Ejecución

```
$ ./interpreter.exe test.txt

stmtlist --> stmtlist NEWLINE
exp --> NUMBER
exp --> PLUS exp
exp --> NUMBER
exp --> MINUS exp
exp --> NUMBER
exp --> exp PLUS exp
exp --> LPAREN exp RPAREN
exp --> exp MULTIPLICATION exp
stmtlist --> stmtlist exp NEWLINE
Correct expression
```

Reconocimiento de operadores unarios

Novedades

Se muestran los errores léxicos

- Funcionamiento

test-error.txt

```
2 * * 4
```

```
dato
```

```
...
```

Ejecución

```
$ ./interpreter.exe test-error.txt
```

```
stmtlist --> epsilon
```

```
exp --> NUMBER
```

```
Program: ./interpreter.exe
```

```
Error line 1 --> Parsing error
```

```
syntax error, unexpected MULTIPLICATION,  
expecting NUMBER or PLUS or MINUS or LPAREN
```

```
stmtlist --> stmtlist error NEWLINE
```

```
stmtlist --> stmtlist NEWLINE
```

```
Program: ./interpreter.exe
```

```
Error line 3 --> Lexical error
```

```
dato
```

```
...
```


Contenido del ejemplo

- 3 Reconocimiento de operadores unarios
 - Novedades
 - Ficheros modificados
 - Ejercicios

Reconocimiento de operadores unarios

Ficheros modificados

Ficheros modificados

- **interpreter.l**
 - **Reconocimiento** de componentes léxicos o tokens con nombre.
 - Uso del estado de flex **ERROR** para controlar componentes léxicos no reconocidos (todavía): identificadores, etc.
- **interpreter.y**
 - Componentes léxicos o tokens con nombre: definición y uso.
 - Reglas para los operadores **unarios**.

Reconocimiento de operadores unarios

Ficheros modificados

interpreter.l (1/2)

- **Reconocimiento** de componentes léxicos o tokens con nombre.

interpreter.l

```
\n  { /* Line counter */
    lineNumber++;
    return NEWLINE;
  }

{NUMBER1}|{NUMBER2} { return NUMBER; }

"-"          { return MINUS; }
"+"          { return PLUS; }
"*"          { return MULTIPLICATION; }
"/"          { return DIVISION; }
"("          { return LPAREN; }
")"          { return RPAREN; }
```

Reconocimiento de operadores unarios

Ficheros modificados

interpreter.l (2/2)

- Uso del estado de flex **ERROR** para controlar componentes léxicos no reconocidos (todavía): identificadores, etc.

interpreter.l

```
/* STATE */
%x ERROR
...
{ /* Any other character */
  BEGIN(ERROR);
  yymore();
}

<ERROR>[~0-9+~*/() \t\n] { yymore(); }
<ERROR>(.\|\\n)          {
  yyless(yytext-1);
  warning("Lexical error", yytext);
  BEGIN(INITIAL);
}
```

Reconocimiento de operadores unarios

Ficheros modificados

interpreter.y (1/3)

- Componentes léxicos o tokens con nombre: definición

interpreter.y

```
%token NEWLINE
%token NUMBER

/* Left associativity */
%left PLUS MINUS
%left MULTIPLICATION DIVISION
%left LPAREN RPAREN

/* Maximum precedence */
%nonassoc UNARY
```

Reconocimiento de operadores unarios

Ficheros modificados

interpreter.y (2/3)

- Componentes léxicos o tokens con nombre usados en las reglas gramaticales

interpreter.y

```
stmtlist: /* empty: epsilon rule */
        | stmtlist NEWLINE /* empty line */
        | stmtlist exp NEWLINE
        | stmtlist error NEWLINE
        ;
```

Reconocimiento de operadores unarios

Ficheros modificados

interpreter.y (3/3)

- Componentes léxicos o tokens con nombre usados en las reglas gramaticales

interpreter.y

```
exp: NUMBER
| exp PLUS exp
| exp MINUS exp
| exp MULTIPLICATION exp
| exp DIVISION exp
| LPAREN exp RPAREN
| PLUS exp
| MINUS exp
;
```

Contenido del ejemplo

- 3 Reconocimiento de operadores unarios
 - Novedades
 - Ficheros modificados
 - Ejercicios

Reconocimiento de operadores unarios

Ejercicios

Ejercicio (Cambio de operadores)

- Cambiar el **operador aritmético de la suma**:
 - $2 + 3 \Rightarrow 2 \& 3$
- Cambiar el **operador aritmético de la multiplicación**:
 - $2 * 3 \Rightarrow 2 \# 3$

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas**
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento-reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 4 Evaluación de expresiones aritméticas
 - Novedades
 - Ficheros modificados

Evaluación de expresiones aritméticas

Novedades

Novedades

- Se **evalúan** las expresiones aritméticas compuestas por números y se muestra el resultado

Ejemplo

```
$/interpreter.exe
```

```
2 + 3
```

```
Result: 5
```

```
5 * 4
```

```
Result: 20
```

Contenido del ejemplo

- 4 Evaluación de expresiones aritméticas
 - Novedades
 - Ficheros modificados

Evaluación de expresiones aritméticas

Ficheros modificados

Ficheros modificados

- **interpreter.l**
 - El analizador léxico **comunica** el valor del número al analizador sintáctico
- **interpreter.y**
 - Utiliza el valor de los números para **evaluar** las expresiones.

Evaluación de expresiones aritméticas

Ficheros modificados

interpreter.l

- El analizador léxico **comunica** el valor del número al analizador sintáctico.

interpreter.l

```
{NUMBER1}|{NUMBER2} {  
    /* Conversion of type and sending  
    of the numerical value to the parser */  
    yyval.number = atof(yytext);  
    return NUMBER;  
}
```

Evaluación de expresiones aritméticas

Ficheros modificados

interpreter.y

- Utiliza el valor de los números para **evaluar** las expresiones.

interpreter.y

```

/* Data type YYSTYPE */
%union
{
  double number;
}
/* Data type of the non-terminal symbol "exp" */
%type <number> exp
...
exp: NUMBER                { $$ = $1; }
  | exp PLUS exp           { $$ = $1 + $3; }
  | exp MINUS exp          { $$ = $1 - $3; }
  | exp MULTIPLICATION exp { $$ = $1 * $3; }
  | exp DIVISION exp       { $$ = $1 / $3; }
  | LPAREN exp RPAREN      { $$ = $2; }
  | PLUS exp %prec UNARY   { $$ = + $2; }
  | MINUS exp %prec UNARY  { $$ = - $2; }
;

```


Evaluación de expresiones aritméticas

Ficheros modificados

interpreter.y

• Glosario

- **yyval**: atributo de un componente léxico.
- **YYSTYPE**: tipo de dato del atributo (véase el fichero interpreter.tab.h).
- **\$\$**: atributo del símbolo **no terminal** de la **parte izquierda** de la regla.
- **\$1**: atributo del **primer** símbolo de la parte derecha de la regla.
- **\$2**: atributo del **segundo** símbolo de la parte derecha de la regla.
- **\$n**: atributo del símbolo **n-ésimo** de la parte derecha de la regla.

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores**
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento-reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 5 Separador de expresiones y nuevos operadores
 - Novedades
 - Ficheros modificados

Separador de expresiones y nuevos operadores

Novedades

Novedades

- **Separador de expresiones**

- Se utiliza el símbolo “;” para **separar** expresiones aritméticas.

- **Nuevos operadores**

- **Resto de la división entera:** % (asociatividad por la **izquierda**).
- **Potencia:** ^ (asociatividad por la **derecha**).

Separador de expresiones y nuevos operadores

Novedades

Separador de expresiones

- Se utiliza el símbolo “;” para **separar** expresiones aritméticas.

Ejemplo

```
$/interpreter.exe
```

```
2+3; 4*5;
```

```
Result: 5
```

```
Result: 20
```

Separador de expresiones y nuevos operadores

Novedades

Nuevos operadores

- **Resto de la división entera:** `%` (asociatividad por la izquierda)

Ejemplo

```
$/interpreter.exe  
8%3;  
Result: 2  
  
20%7%5;  
Result: 1  
  
(20%7)%5;  
Result: 1  
  
20%(7%5);  
Result: 0
```

Separador de expresiones y nuevos operadores

Novedades

Nuevos operadores

- **Potencia:** \wedge (asociatividad por la derecha)

Ejemplo

```
$/interpreter.exe
```

```
2^3;
```

```
Result: 8
```

```
2^3^2;
```

```
Result: 512
```

```
(2^3)^2;
```

```
Result: 64
```

Contenido del ejemplo

- 5 Separador de expresiones y nuevos operadores
 - Novedades
 - Ficheros modificados

Separador de expresiones y nuevos operadores

Ficheros modificados

Ficheros modificados

- **interpreter.l**

- Reconomiento de nuevos componentes léxicos:
SEMICOLON, MODULO, POWER.
- Al reconocer `'\n'`, **no** se devuelve **NEWLINE** pero se incrementa el contador de líneas.

- **interpreter.y**

- Definición de nuevos componentes léxicos:
SEMICOLON, MODULO, POWER.
- Regla gramatical para la división entera.
- Regla gramatical para la potencia.

Separador de expresiones y nuevos operadores

Ficheros modificados

interpreter.l (1/2)

- Reconomiento de nuevos componentes léxicos:
SEMICOLON, MODULO, POWER.

interpreter.l

```
";" { return SEMICOLON; }
```

```
...
```

```
"%" { return MODULO; }
```

```
"^" { return POWER; }
```

Separador de expresiones y nuevos operadores

Ficheros modificados

interpreter.l (2/2)

- Al reconocer `'\n'`, **no** se devuelve **NEWLINE** pero se incrementa el contador de líneas.

interpreter.l

```
\n    {  
        /* Line counter */  
        lineNumber++;  
    }
```

Separador de expresiones y nuevos operadores

Ficheros modificados

interpreter.y (1/2)

- Definición de nuevos componentes léxicos:
SEMICOLON, MODULO, POWER.

interpreter.y

```
/* Minimum precedence */
%token SEMICOLON
...
%left MULTIPLICATION DIVISION MODULO
...
/* Maximum precedence */
%right POWER
```

Separador de expresiones y nuevos operadores

Ficheros modificados

interpreter.y (2/2)

- Regla gramatical para el **resto de la división entera**.
- Regla gramatical para la **potencia**.

interpreter.y

```
exp: ...  
  | exp MODULO exp { $$ = (int) $1 % (int) $3; }  
  | exp POWER exp { $$ = pow($1,$3); }  
  ;
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución**
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 6 Recuperación de errores de ejecución
 - Novedades
 - Ficheros nuevos y modificados

Recuperación de errores de ejecución

Novedades

Recuperación de errores de ejecución

- Se **detecta** el **error**, pero **no termina la ejecución**.
- Se controla la **división por cero**.

Ejemplo

```
$/interpreter.exe
8%0;
Program: ./interpreter.exe
Error line 1 --> Runtime error in modulo
Division by zero

3/0;
Program: ./interpreter.exe
Error line 2 --> Runtime error in division
Division by zero
```


Contenido del ejemplo

- 6 Recuperación de errores de ejecución
 - Novedades
 - Ficheros nuevos y modificados

Recuperación de errores de ejecución

Ficheros nuevos y modificados

Ficheros nuevos del subdirectorio /error

- **error.hpp**: prototipo de funciones de recuperación de errores
- **error.cpp**: funciones de recuperación de errores

Recuperación de errores de ejecución

Ficheros nuevos y modificados

error.hpp

- Prototipo de funciones de recuperación de errores

error.hpp

```
void execerror(std::string errorMessage1,  
               std::string errorMessage2);  
  
void fpecatch(int signum);
```

Recuperación de errores de ejecución

Ficheros nuevos y modificados

error.cpp

- Funciones de recuperación de errores

error.cpp

```
void execerror(std::string errorMessage1,
               std::string errorMessage2)
{
    warning(errorMessage1,errorMessage2);

    longjmp(begin,0); /* return to a viable state */
}

void fpecatch(int signum)
{
    execerror("Runtime","floating point error");
}
```

Recuperación de errores de ejecución

Ficheros nuevos y modificados

Ficheros modificados

- **interpreter.cpp**
 - Se establece un **estado viable** de recuperación de **errores**.
- **interpreter.y:**
 - Declara la variable para la recuperación de errores
 - Acciones semánticas para el control de errores

Recuperación de errores de ejecución

Ficheros nuevos y modificados

interpreter.cpp

- Se establece un **estado viable** de recuperación de **errores**.

interpreter.cpp

```
// Use for recovery of runtime errors
#include <setjmp.h>
#include <signal.h>
...
extern jmp_buf begin; //!< It enables recovery of runtime errors
...
/* A viable state to continue after a runtime error is set */
setjmp(begin);

/* The name of the function to handle floating-point errors is set */
signal(SIGFPE, fpecatch);
```

Recuperación de errores de ejecución

Ficheros nuevos y modificados

interpreter.y (1/2)

- Declara la variable para la recuperación de errores

interpreter.y

```
/* Use for recovery of runtime errors */
#include <setjmp.h>
#include <signal.h>

/* Error recovery functions */
#include "../error/error.hpp"
...
jmp_buf begin; //!< It enables recovery of runtime errors
```

Recuperación de errores de ejecución

Ficheros nuevos y modificados

interpreter.y (2/2)

- Acciones semánticas para el control de errores

interpreter.y

```
#define ERROR_BOUND 1.0e-6 //!< To compare real numbers
...
exp: ...
  | exp DIVISION exp
  {
    if (fabs($3) < ERROR_BOUND)
      execerror("Runtime error in division", "Division by zero")
    else
      $$ = $1 / $3;
  }
  | exp MODULO exp
  {
    if (fabs($3) < ERROR_BOUND)
      execerror("Runtime error in modulo", "Division by zero");
    else
      $$ = (int) $1 % (int) $3;
  }
}
```


Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos**
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

7 Variables y tabla de símbolos

- Novedades
- Ficheros nuevos y modificados
- Ejercicio

Variables y tabla de símbolos

Novedades

Novedades

- **Variables numéricas de tipo real.**
- Sentencia de **asignación múltiple.**
- **Tabla de símbolos.**

Variables y tabla de símbolos

Novedades

Variables numéricas de tipo real

- Empiezan por una **letra** que puede ir **seguida** de más **letras** o **dígitos**.
 - a, dato, iva, x1, punto32, ...
- Se almacenan en la **tabla de símbolos** (map de STL)

Variables no admitidas

- 1dato, iva_1, \$3, ...

Variables y tabla de símbolos

Novedades

Variables numéricas de tipo real

```
dato = 3;
```

```
Result: 3
```

```
dato;
```

```
Result: 3
```

```
dato = 2 * dato;
```

```
Result: 6
```

```
dato;
```

```
Result: 6
```

Variables y tabla de símbolos

Novedades

Variables numéricas de tipo real

- Se controlan las variables **no** definidas.

Ejemplo

```
2+iva;
```

```
Program: ./interpreter.exe
```

```
Error line 1 --> The variable is UNDEFINED
```

```
iva
```

Variables y tabla de símbolos

Novedades

Sentencia de asignación múltiple

```
a = b = c = 7;
```

```
Result: 7
```

```
a; b; c;
```

```
Result: 7
```

```
Result: 7
```

```
Result: 7
```

Variables y tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- **TableInterface.**
- **Table.**

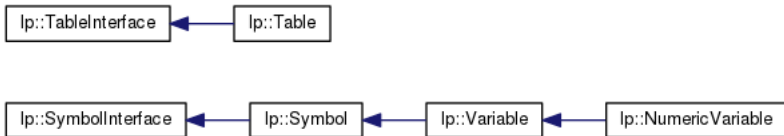
- **SymbolInterface.**
- **Symbol.**
- **Variable.**
- **NumericVariable.**

Variables y tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- Jerarquía de clases

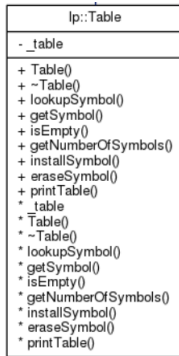
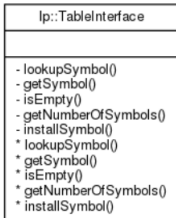


Variables y tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- Clases **TableInterface** y **Table**.

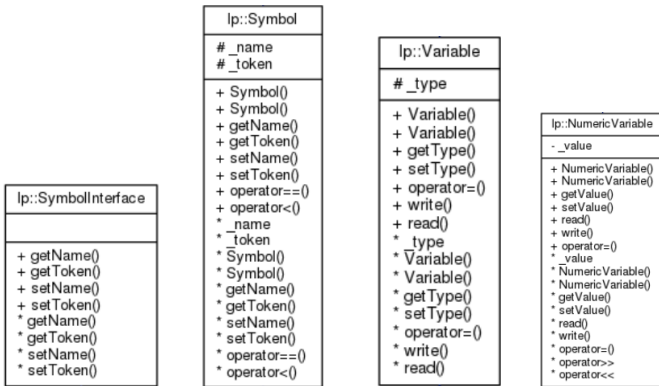


Variables y tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- Clases **SymbolInterface**, **Symbol**, **Variable** y **NumericVariable**.



Contenido del ejemplo

- 7 Variables y tabla de símbolos
 - Novedades
 - Ficheros nuevos y modificados
 - Ejercicio

Variables y tabla de símbolos

Ficheros nuevos y modificados

Ficheros nuevos (1/2)

- **makefile** del subdirectorio **table**.
- **tableInterface.hpp**: definición de la clase abstracta **TableInterface**.
- **table.hpp**: definición de la clase **Table**
- **table.cpp**: código del resto de funciones de la clase **Table**.

Variables y tabla de símbolos

Ficheros nuevos y modificados

Ficheros nuevos (2/2)

- **symbolInterface.hpp**: definición de la clase abstracta **SymbolInterface**.
- **symbol.hpp**: definición de la clase **Symbol**.
- **symbol.cpp**: código del resto de funciones de la clase **Symbol**.
- **variable.hpp**: definición de la clase **Variable**, que hereda de **Symbol**
- **variable.cpp**: código del resto de funciones de la clase **Variable**.
- **numericVariable.hpp**: definición de la clase **NumericVariable**, que hereda de **Variable**
- **numericVariable.cpp**: código del resto de funciones de la clase **NumericVariable**.

Variables y tabla de símbolos

Ficheros nuevos y modificados

Ficheros modificados

- **interpreter.l:**
 - Definiciones regulares.
 - Reconocimiento de variables e instalación en la tabla de símbolos
- **interpreter.y:**
 - Definición de componentes léxicos: **VARIABLE**, **UNDEFINED**.
 - Reglas gramaticales para el uso de variables.
- **Doxyfile:** generación de la documentación del subdirectorio **table**.
- **makefile principal:** referencia a los objetos del subdirectorio **table** y acceso al **makefile** del subdirectorio **table**.
- **makefile** del subdirectorio **table:** compilación de los ficheros de la tabla de símbolos.

Variables y tabla de símbolos

Ficheros nuevos y modificados

interpreter.l (1/2)

- Definiciones regulares.

interpreter.l

```
DIGIT      [0-9]
LETTER     [a-zA-Z]

NUMBER1    {DIGIT}+\.?
NUMBER2    {DIGIT}*\.{DIGIT}+

IDENTIFIER {LETTER}({LETTER}|{DIGIT})*
```


Variables y tabla de símbolos

Ficheros nuevos y modificados

interpreter.l (2/2)

- **Reconocimiento** de variables e **instalación** en la tabla de símbolos.

interpreter.l

```
{IDENTIFIER} {
    std::string identifier(yytext);
    ylval.identifier = strdup(yytext);

    if (table.lookupSymbol(identifier) == false)
    {
        lp::Variable *var = new lp::Variable(identifier, VARIABLE, UNDEFINED);
        table.installSymbol(var);
    }

    /* The identifier is returned as VARIABLE */
    return VARIABLE;
}
```

Variables y tabla de símbolos

Ficheros nuevos y modificados

interpreter.y (1/2)

- Definición de componentes léxicos: **VARIABLE, UNDEFINED.**

interpreter.y

```
/* Data type YYSTYPE */
%union {
    double number;
    char * identifier;
}
...
%token <identifier> VARIABLE UNDEFINED
```

Variables y tabla de símbolos

Ficheros nuevos y modificados

interpreter.y (1/2)

- Reglas gramaticales para el uso de variables.

interpreter.y

```
exp: ...  
    | VARIABLE ASSIGNMENT exp    { ... }  
    | VARIABLE                    { ... }  
    ;
```

Variables y tabla de símbolos

Ficheros nuevos y modificados

Doxyfile

- Generación de la documentación del subdirectorio **table**.

Doxyfile

```
INPUT = interpreter.cpp parser error includes table
```

Variables y tabla de símbolos

Ficheros nuevos y modificados

makefile principal

- Referencia a los objetos del subdirectorio **table**.
- Acceso al makefile del subdirectorio **table**.

makefile principal

```
OBJECTS-TABLE = table/*.o #New in example 7

$(NAME).exe: parser-dir error-dir table-dir $(OBJECTS)
    @echo "Generating " $@
    @$(CPP) $(OBJECTS) $(OBJECTS-PARSER) $(OBJECTS-ERROR) $(OBJECTS-TABLE) $(LFLAGS) -o $@
...
table-dir:
    @echo "Accessing directory table"
    @echo
    @make -C table/
    @echo
```

Variables y tabla de símbolos

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los ficheros de la tabla de símbolos.

makefile del subdirectorio table

```
NAME=table
...
OBJECTS= $(NAME).o symbol.o variable.o numericVariable.o
...
# Main rule
all: $(OBJECTS)
$(NAME).o: $(NAME).cpp $(NAME).hpp tableInterface.hpp symbol.hpp symbolInterface.hpp
    @echo "Compiling " $<
    @$(CPP) $(CFLAGS) $<

symbol.o: ...

variable.o: ...

numericVariable.o: numericVariable.cpp numericVariable.hpp variable.hpp symbol.hpp symbolInterface.hpp
    @echo "Compiling " $<
    @$(CPP) $(CFLAGS) $<
...
```

Contenido del ejemplo

- 7 Variables y tabla de símbolos
 - Novedades
 - Ficheros nuevos y modificados
 - Ejercicio

Variables y tabla de símbolos

Ejercicio

Ejercicio

- *Utiliza el operador de asignación de Pascal*

```
dato := 3;
```


Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción**
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 8 Conflicto de desplazamiento-reducción
 - Novedades
 - Ficheros nuevos y modificados

Conflicto de desplazamiento-reducción

Novedades

Novedades

- Gramática que genera un **conflicto de desplazamiento - reducción**.
- El conflicto será corregido en el ejemplo 9.

Conflicto de desplazamiento - reducción

```
$ make
Accessing directory parser
make[1]: se entra en el directorio
'../ejemplo8/parser'
Generando: interpreter.tab.c interpreter.tab.h
interpreter.y: aviso: 1 conflicto desplazamiento/reducción [-Wconflicts-sr]
```

Conflicto de desplazamiento - reducción

- La sentencia de **asignación** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
    
```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
    
```

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
    
```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
    
```

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```


Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;
    
```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;
    
```

Primera derivación por la derecha en orden inverso

program

- ⇒ stmtlist
 - ⇒ stmtlist asgn SEMICOLON
 - ⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
 - ⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
 - ⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
 - ⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
- dato = 3 ;

Segunda derivación por la derecha en orden inverso

program

- ⇒ stmtlist
 - ⇒ stmtlist exp SEMICOLON
 - ⇒ stmtlist asgn SEMICOLON
 - ⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
 - ⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
 - ⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
 - ⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
- dato = 3 ;

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Primera derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Segunda derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist exp SEMICOLON
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato          =          3          ;

```

Contenido del ejemplo

- 8 Conflicto de desplazamiento-reducción
 - Novedades
 - Ficheros nuevos y modificados

Conflicto de desplazamiento-reducción

Ficheros nuevos y modificados

interpreter.output

- Fichero que describe la **tabla LALR** y el **conflicto de desplazamiento - reducción**.

interpreter.output

```
Estado 11 conflictos: 1 desplazamiento/reducción
...
Estado 11
4 stmtlist: stmtlist asgn . SEMICOLON
18 exp: asgn .

SEMICOLON  desplazar e ir al estado 19

SEMICOLON  [reduce usando la regla 18 (exp)]
$default   reduce usando la regla 18 (exp)
...
```

Conflicto de desplazamiento-reducción

Ficheros nuevos y modificados

interpreter.y

- Uso del nuevo símbolo no terminal `asgn`.

interpreter.y

```
%type <identifier> asgn
...
stmtlist:
    ...
    | stmtlist asgn SEMICOLON
    ...
    ;
...
asgn: VARIABLE ASSIGNMENT exp
    ;
...
exp: ...
    | asgn
    ...
    ;
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
 - **Novedades**
 - Ficheros modificados
 - Ejercicios

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8.
- **Lectura de variables.**
- **Escritura de los valores** de las expresiones aritméticas.

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
  
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
  
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
  
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
  
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;

```


Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- Se **resuelve** el **conflicto de desplazamiento – reducción** del ejemplo 8: véase el fichero **interpreter.y**.

Única derivación por la derecha en orden inverso

```

program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist asgn SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ε VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ VARIABLE ASSIGNMENT NUMBER SEMICOLON
dato           =           3           ;

```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Novedades

Novedades

- **Lectura de variables:** sentencia **read**.
- **Escritura de los valores** de las expresiones aritméticas: sentencia **print**.

Ejemplo

```
$ ./interpreter.exe
read(dato);
Insert a numeric value --> 9
print dato;
Print: 9

print 2 * dato;
Print: 18
```

Contenido del ejemplo

- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
 - Novedades
 - Ficheros modificados
 - Ejercicios

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ficheros modificados

Ficheros modificados

- **interpreter.l**
 - Reconocimiento de los símbolos terminales **READ** y **PRINT**.
- **interpreter.y**
 - Definición de los símbolos **terminales READ** y **PRINT**.
 - Nuevos símbolos **no terminales read** y **print** y sus reglas.
 - Nuevo símbolo **no terminal stmt** y sus reglas.
 - **Modificación** de las reglas gramaticales de **stmtlist**.
 - **Supresión** de la regla **exp** → **asgn**.
 - Nueva regla de **asgn** para la **asignación múltiple**.

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ficheros modificados

interpreter.l

- Reconocimiento de los símbolos **terminales READ y PRINT**.

interpreter.l

```
print    {return PRINT;}
```

```
read    {return READ;}
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ficheros modificados

interpreter.y (1/3)

- Definición de los símbolos **no terminales** **READ** y **PRINT**.
- Nuevos símbolos **no terminales** **read** y **print** y sus reglas.

interpreter.y

```
%token    PRINT READ
...
print:    PRINT exp    ...
        ;
read:     READ LPAREN VARIABLE RPAREN    ...
        ;
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ficheros modificados

interpreter.y (2/3)

- Nuevo símbolo **no terminal** `stmt` y sus reglas.
- **Modificación** de las reglas gramaticales de `stmtlist`.

interpreter.y

```
stmtlist: /* empty: epsilon rule */
  | stmtlist stmt
  | stmtlist error
  ;

stmt: SEMICOLON /* Empty statement: ";" */
  | asgn SEMICOLON
  | print SEMICOLON
  | read SEMICOLON
  ;
```

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ficheros modificados

interpreter.y (3/3)

- Supresión de la regla `exp` \rightarrow `asgn`.
- Nueva regla de `asgn` para la asignación múltiple.

interpreter.y

```
asgn: VARIABLE ASSIGNMENT exp
    | VARIABLE ASSIGNMENT asgn
    ;
```


Contenido del ejemplo

- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
 - Novedades
 - Ficheros modificados
 - Ejercicios

Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura

Ejercicios

Ejercicios

- *Definición regular para **no distinguir** mayúsculas de minúsculas en print*

```
(?:i:print) return PRINT;
```

- **No distinguir** mayúsculas de minúsculas en los identificadores

```
for (int i = 0; yytext[i] != '\0'; i++)  
{  
    yytext[i] = toupper(yytext[i]);  
}
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 **Constantes predefinidas que se pueden modificar**
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 10 Constantes predefinidas que se pueden modificar
 - Novedades
 - Ficheros nuevos y modificados

Constantes predefinidas que se pueden modificar

Novedades

Se permite el uso de constantes predefinidas

- **PI:** 3.14159265358979323846

$$\pi = \frac{\text{circunferencia}}{\text{radio}}$$

- **E:** 2.71828182845904523536

$$\text{Base natural: } e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots$$

- **GAMMA:** 0.57721566490153286060

$$\text{Constante de Euler-Mascheroni: } \gamma = \lim_{n \rightarrow \infty} \left(-\ln(n) + \sum_{k=1}^n \frac{1}{k} \right)$$

- **DEG:** 57.29577951308232087680

$$\text{Grado por radián} = \frac{180}{\pi}:$$

- **PHI:** 1.61803398874989484820

$$\text{Proporción áurea: } \phi = \frac{\sqrt{5}+1}{2}$$

Constantes predefinidas que se pueden modificar

Novedades

Ejemplo

```
$ ./interpreter.exe  
print PI;  
Print: 3.141593  
  
print E;  
Print: 2.718282  
  
print GAMMA;  
Print: 0.5772157  
  
print DEG;  
Print: 57.29578  
  
print PHI;  
Print: 1.618034
```

Constantes predefinidas que se pueden modificar

Novedades

Las constantes se pueden modificar

```
$ ./interpreter.exe  
PI = 3;  
print PI;  
Print: 3
```

Constantes predefinidas que se pueden modificar

Novedades

Las constantes se pueden modificar

```
$ ./interpreter.exe  
PI = 3;  
print PI;  
Print: 3
```

Nota

El ejemplo 11 **evitará** modificar las constantes predefinidas.

Contenido del ejemplo

- 10 Constantes predefinidas que se pueden modificar
 - Novedades
 - Ficheros nuevos y modificados

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

Ficheros nuevos

- **init.hpp**
 - Definición de las **constantes predefinidas**.
 - Prototipo de la función **init**.
- **init.cpp**
 - Código de la función **init** que inicializa la tabla de símbolos con las constantes predefinidas.

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

init.hpp

- Definición de las **constantes predefinidas**.
- Prototipo de la función **init**.

init.hpp

```
static struct {
    std::string name;
    double value;
} numericConstant[] = {
    {"PI",      3.14159265358979323846},
    {"E",       2.71828182845904523536},
    {"GAMMA",   0.57721566490153286060},
    {"DEG",     57.29577951308232087680},
    {"PHI",     1.61803398874989484820},
    {"",        0}
};

void init(lp::Table &t);
```

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

init.cpp

- Código de la función **init** que inicializa la tabla de símbolos con las constantes predefinidas.

init.cpp

```
void init(lp::Table &t)
{
    int i;
    lp:: NumericVariable *n;
    // The predefined numeric constants are installed in the table of symbols
    for (i=0; numericConstant[i].name.compare("")!=0; i++)
    { // The predefined numeric constant is inserted into the symbol table
        n = new lp::NumericVariable(numericConstant[i].name,
                                    VARIABLE,
                                    NUMBER,
                                    numericConstant[i].value);

        // A pointer to the new NumericVariable is inserted into the table of symbols
        t.installSymbol(n);
    }
}
```

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

Ficheros modificados

- **interpreter.cpp**
 - Llamada a **init(table)**, que inicializa la tabla de símbolos con las constantes predefinidas.
- **interpreter.y**
 - Inclusión del fichero de cabecera **init.hpp**
- **makefile** del subdirectorio **table**:
 - Compilación de los nuevos ficheros **init.hpp** e **init.cpp**.

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

interpreter.cpp

- Llamada a **init(table)**, que inicializa la tabla de símbolos con las constantes predefinidas.

interpreter.cpp

```
int main(int argc, char *argv[])
{
    ...
    init(table);
    ...
    /* Parser function */
    yyparse();

    /* End of program */
    return 0;
}
```

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

interpreter.y

- Inclusión del fichero de cabecera **init.hpp**

interpreter.cpp

```
...  
#include "../table/init.hpp"  
...
```

Constantes predefinidas que se pueden modificar

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los nuevos ficheros **init.hpp** e **init.cpp**.

makefile

```
OBJECTS= $(NAME).o symbol.o variable.o numericVariable.o \  
         init.o  
...  
init.o: init.cpp numericVariable.hpp variable.hpp symbol.hpp symbolInterface.hpp $(NAME).hpp \  
        tableInterface.hpp ../parser/interpreter.tab.h  
@echo "Compiling " $<  
@$(CPP) $(CFLAGS) $<
```


Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar**
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 11 Constantes predefinidas que no se pueden modificar
 - Novedades
 - Ficheros nuevos y modificados

Constantes predefinidas que no se pueden modificar

Novedades

Constantes predefinidas

- **No se pueden modificar** en las sentencias de asignación o lectura.
- Se utilizan **reglas gramaticales** para controlar los **errores**.
- Nuevas clases: **Constant** y **NumericConstant**.

Constantes predefinidas que no se pueden modificar

Novedades

Constantes predefinidas

- **No se pueden modificar** en las sentencias de asignación o lectura.
- Se utilizan **reglas gramaticales** para controlar los **errores**.

Ejemplo

```
PI = 3;
Program: ./interpreter.exe
Error line 1 --> Semantic error in assignment: it is not allowed to modify a constant
  PI

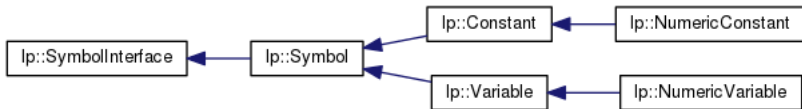
read(PI);
Program: ./interpreter.exe
Error line 3 --> Semantic error in "read statement": it is not allowed to modify a constant
  PI
```

Constantes predefinidas que no se pueden modificar

Novedades

Clases de la Tabla de símbolos

- Nuevas clases: **Constant** y **NumericConstant**.



Constantes predefinidas que no se pueden modificar

Novedades

Clases de la Tabla de símbolos

- Nuevas clases: **Constant** y **NumericConstant**.

lp::Constant
#_type
+ Constant()
+ Constant()
+ getType()
+ setType()
+ operator=()
+ write()
+ read()
*_type
* Constant()
* Constant()
* getType()
* setType()
* operator=()
* write()
* read()

lp::NumericConstant
-_value
+ NumericConstant()
+ NumericConstant()
+ getValue()
+ setValue()
+ read()
+ write()
+ operator=()
*_value
* NumericConstant()
* NumericConstant()
* getValue()
* setValue()
* read()
* write()
* operator=()
* operator>>
* operator<<

Contenido del ejemplo

- 11 Constantes predefinidas que no se pueden modificar
 - Novedades
 - Ficheros nuevos y modificados

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

Ficheros nuevos

- **constant.hpp**
 - Definición de la clase **Constant**, que hereda de la clase **Symbol**.
- **constant.cpp**
 - Código del resto de funciones de la clase **Constant**.
- **numericConstant.hpp**
 - Definición de la clase **NumericConstant**, que hereda de **Constant**.
- **numericConstant.cpp**
 - Código del resto de funciones de la clase **NumericConstant**.

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

Ficheros modificados

- **init.cpp**
 - Se instalan las constantes predefinidas en la tabla de símbolos con el token **CONSTANT** y el tipo **NUMBER**.
- **interpreter.l**
 - Si un identificador está instalado en la tabla de símbolos, se devuelve su token o componente léxico: **VARIABLE** o **CONSTANT**.
- **interpreter.y**
 - Reglas gramaticales de **control de errores**.
- **makefile** del subdirectorio **table**: compilación de los nuevos ficheros.
 - constant.hpp, constant.cpp
 - numericConstant.hpp, numericConstant.cpp

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

init.cpp

- Se instalan las constantes predefinidas en la tabla de símbolos con el token **CONSTANT** y el tipo **NUMBER**.

init.cpp

```
void init(lp::Table &t)
{
    int i;
    lp::NumericConstant *n;
    for (i=0; numericConstant[i].name.compare("")!=0; i++)
    { // The predefined numeric constant is inserted into the symbol table
        n = new lp::NumericConstant(numericConstant[i].name,
                                    CONSTANT,
                                    NUMBER,
                                    numericConstant[i].value);

        // A pointer to the new NumericConstant is inserted into the table of symbols
        t.installSymbol(n);
    }
}
```

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

interpreter.l

- Si un identificador está instalado en la tabla de símbolos, se devuelve su token o componente léxico.

interpreter.l

```
{IDENTIFIER} { std::string identifier(yytext);
                ylval.identifier = strdup(yytext);

                if (table.lookupSymbol(identifier) == false)
                {
                    lp::Variable *var = new lp::Variable(identifier, VARIABLE, UNDEFINED);
                    table.installSymbol(var);
                    return VARIABLE;
                }
                else
                {
                    lp::Symbol *s = table.getSymbol(identifier);
                    return s->getToken();
                }
            }
```

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

interpreter.y

- Reglas gramaticales de **control de errores**.

interpreter.y

```
read: ...
| READ LPAREN CONSTANT RPAREN
  { execerror("Semantic error in \"read statement\": it is not allowed to modify a constant ",$3); }
...

asgn: ...
| CONSTANT ASSIGNMENT exp
  { execerror("Semantic error in assignment: it is not allowed to modify a constant ", $1); }
| CONSTANT ASSIGNMENT asgn
  { execerror("Semantic error in multiple assignment: it is not allowed to modify a constant ",$1); }
;
```

Constantes predefinidas que no se pueden modificar

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los nuevos ficheros:
 - constant.hpp, constant.cpp
 - numericConstant.hpp, numericConstant.cpp

makefile

```
OBJECTS= table.o symbol.o variable.o numericVariable.o \  
        init.o \  
        constant.o numericConstant.o  
...  
constant.o: constant.cpp constant.hpp symbol.hpp symbolInterface.hpp  
    @echo "Compiling " $<  
    @$(CPP) $(CFLAGS) $<  
  
numericConstant.o: numericConstant.cpp numericConstant.hpp constant.hpp symbol.hpp symbolInterface.hpp  
    @echo "Compiling " $<  
    @$(CPP) $(CFLAGS) $<  
...
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12** Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 12 Palabras claves pre-instaladas en la tabla de símbolos
 - Novedades
 - Ficheros nuevos y modificados

Palabras claves pre-instaladas en la tabla de símbolos

Novedades

Palabras claves

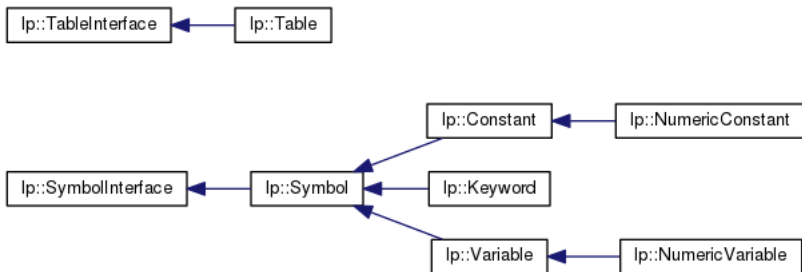
- Son **pre-instaladas** en la tabla de símbolos
- Nueva clase de la Tabla de Símbolos: **Keyword**.

Palabras claves pre-instaladas en la tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- Nueva clase: **Keyword**.



Palabras claves pre-instaladas en la tabla de símbolos

Novedades

Clases de la Tabla de símbolos

- Nueva clase: **Keyword**.

lp::Keyword
+ Keyword() + Keyword() + operator=() + write() + read() * Keyword() * Keyword() * operator=() * write() * read()

Contenido del ejemplo

- 12 Palabras claves pre-instaladas en la tabla de símbolos
 - Novedades
 - Ficheros nuevos y modificados

Palabras claves pre-instaladas en la tabla de símbolos

Ficheros nuevos y modificados

Ficheros nuevos

- **keyword.hpp**
 - Definición de la clase **Keyword**, que hereda de la clase **Symbol**.
- **keyword.cpp**
 - Código del resto de funciones de la clase **Keyword**.

Palabras claves pre-instaladas en la tabla de símbolos

Ficheros nuevos y modificados

Ficheros modificados

- **interpreter.l**
 - Las reglas de reconocimiento de palabras claves se han **eliminado** porque las palabras claves son **pre-instaladas** en la tabla de símbolos.
- **init.hpp**
 - Definición de las palabras claves
- **init.cpp**
 - **Instalación** de las palabras claves en la tabla de símbolos.
- **makefile** del subdirectorio **table**: compilación de los nuevos ficheros
 - keyword.hpp, keyword.cpp

Palabras claves pre-instaladas en la tabla de símbolos

Ficheros nuevos y modificados

init.hpp

- Definición de las palabras claves

init.hpp

```
static struct
{
  std::string name;
  int token;
} keyword[] = {
  {"print", PRINT},
  {"read", READ},
  {"", 0}
};
```

Palabras claves pre-instaladas en la tabla de símbolos

Ficheros nuevos y modificados

init.cpp

- **Instalación** de las palabras claves en la tabla de símbolos.

init.cpp

```
void init(lp::Table &t)
{
    ...
    lp::Keyword *k;
    for (i=0; keyword[i].name.compare("")!=0; i++)
    {
        k = new lp::Keyword(keyword[i].name, keyword[i].token);
        t.installSymbol(k);
    }
}
```

Palabras claves pre-instaladas en la tabla de símbolos

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los nuevos ficheros
 - **keyword.hpp, keyword.cpp.**

makefile del subdirectorio table

```
OBJECTS= table.o symbol.o variable.o numericVariable.o \  
init.o \  
constant.o numericConstant.o \  
keyword.o  
...  
keyword.o: keyword.cpp keyword.hpp symbol.hpp symbolInterface.hpp  
@echo "Compiling " $<  
@$(CPP) $(CFLAGS) $<  
...
```


Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas**
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

13 Funciones matemáticas predefinidas

- Novedades
- Ficheros nuevos y modificados
- Ejercicio

Funciones matemáticas predefinidas

Novedades

Funciones matemáticas predefinidas

- **Funciones matemáticas** con un argumento:
 - sin, cos, atan, log, log10, exp, sqrt, integer y abs.
- **Nuevas clases** de la Tabla de Símbolos:
 - Función predefinida: clase **Builtin**.
 - Función predefinida con un argumento: clase **BuiltinParameter1**.
- Función **errcheck** para comprobar si una función matemática genera algún **error** en su dominio o rango.

Funciones matemáticas predefinidas

Novedades

Funciones matemáticas predefinidas con un argumento

- sin, cos, atan, log, log10, exp, sqrt, integer, abs

Ejemplo

```
print sin(PI/2);
```

```
Print: 1
```

```
print sqrt(2);
```

```
Print: 1.414214
```

```
print integer(PI);
```

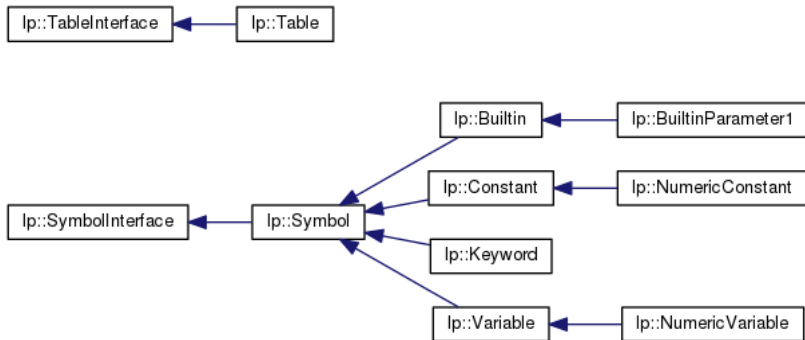
```
Print: 3
```

Funciones matemáticas predefinidas

Novedades

Clases de la Tabla de símbolos (1/2)

- Nuevas clases: **Builtin** y **BuiltinParameter1**.



Funciones matemáticas predefinidas

Novedades

Clases de la Tabla de símbolos (2/2)

- Nuevas clases: **Builtin** y **BuiltinParameter1**.

lp::Builtin
#_nParameters
+ Builtin()
+ Builtin()
+ getNParameters()
+ setNParameters()
+ operator=()
+ write()
+ read()
* _nParameters
* Builtin()
* Builtin()
* getNParameters()
* setNParameters()
* operator=()
* write()
* read()

lp::BuiltinParameter1
- _function
+ BuiltinParameter1()
+ BuiltinParameter1()
+ getFunction()
+ setFunction()
+ operator=()
* _function
* BuiltinParameter1()
* BuiltinParameter1()
* getFunction()
* setFunction()
* operator=()

Funciones matemáticas predefinidas

Novedades

Funciones matemáticas predefinidas

- Función **errcheck** para comprobar si una función matemática genera algún **error** en su dominio o rango.

Ejemplo

```
print log(0);  
Program: ./interpreter.exe  
Error line 1 --> Neperian logarithm  
      Runtime error --> result out of range
```

Contenido del ejemplo

13 Funciones matemáticas predefinidas

- Novedades
- Ficheros nuevos y modificados
- Ejercicio

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros nuevos

- **mathFunction.hpp**
 - Prototipo de las funciones matemáticas predefinidas.
- **mathFunction.cpp**
 - Código de las funciones matemáticas predefinidas.
- **builtin.hpp**
 - Definición de la clase **Builtin**.
- **builtin.cpp**
 - Código de la clase **Builtin**.
- **builtinParameter1.hpp**
 - Definición de la clase **BuiltinParameter1**.
- **builtinParameter1.cpp**
 - Código de la clase **BuiltinParameter1**.

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros modificados (1/2)

- **interpreter.y**
 - Definición del terminal **BUILTIN**.
 - Regla para el uso de funciones matemáticas con un argumento.
- **error.hpp**
 - Prototipo de la nueva función **errcheck**.
- **error.cpp**
 - Código de la nueva función **errcheck**.

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros modificados (2/2)

- **init.hpp**
 - Definición de las funciones matemáticas predefinidas.
- **init.cpp**
 - Instalación en la tabla de símbolos de las funciones matemáticas predefinidas.
- **makefile** del subdirectorio **table**: compilación de los nuevos ficheros
 - mathFunction.hpp, mathFunction.cpp
 - builtin.hpp, builtin.cpp
 - builtinParameter1.hpp, builtinParameter1.cpp

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

interpreter.y

- Definición del terminal **BUILTIN**.
- Regla para el uso de funciones matemáticas con un argumento.

interpreter.y

```
%token <identifier> VARIABLE UNDEFINED CONSTANT BUILTIN
...
exp: ...
    | BUILTIN LPAREN exp RPAREN
    ;
```

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

init.hpp

- Definición de las funciones matemáticas predefinidas.

init.hpp

```
static struct {
    std::string name ;
    lp::TypePointerDoubleFunction_1 function;
} function_1 [] = {
    {"sin",    sin},
    {"cos",    cos},
    {"atan",   atan},
    {"log",    Log},
    {"log10",  Log10},
    {"exp",    Exp},
    {"sqrt",   Sqrt},
    {"integer", integer},
    {"abs",    fabs},
    {"",       0}
};
```

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

builtinParameter1.hpp

- New type definition: **TypePointerDoubleFunction_1**

builtinParameter1.hpp

```
typedef double (*TypePointerDoubleFunction_1)(double x);
```

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

init.cpp

- **Instalación** en la tabla de símbolos de las funciones matemáticas predefinidas

init.cpp

```
void init(lp::Table &t)
{
    ...
    lp::BuiltinParameter1 *f;
    for (i=0; function_1[i].name.compare("")!=0; i++)
    {
        f = new lp::BuiltinParameter1(function_1[i].name,
                                      BUILTIN,    // Token
                                      1,          // Number of parameters
                                      function_1[i].function);

        t.installSymbol(f);
    }
}
```

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

error.hpp

- Prototipo de la nueva función **errcheck**.

error.hpp

```
double errcheck(double d, std::string s);
```


Funciones matemáticas predefinidas

Ficheros nuevos y modificados

error.cpp

- Código de la nueva función **errcheck**.

error.cpp

```
double errcheck(double d, std::string s)
{
    if (errno == EDOM) {
        errno=0;
        std::string msg("Runtime error --> argument out of domain");
        std::cout << msg << std::endl;
        execerror(s,msg);
    }
    else if (errno == ERANGE) {
        errno=0;
        std::string msg("Runtime error --> result out of range");
        execerror(s,msg);
    }
    return d;
}
```

Funciones matemáticas predefinidas

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los nuevos ficheros
 - mathFunction.hpp, mathFunction.cpp
 - builtin.hpp, builtin.cpp
 - builtinParameter1.hpp, builtinParameter1.cpp

makefile

```
OBJECTS= table.o symbol.o variable.o numericVariable.o init.o constant.o numericConstant.o keyword.o \  
        builtin.o builtinParameter1.o mathFunction.o  
    ...  
builtin.o: builtin.cpp builtin.hpp symbol.hpp symbolInterface.hpp  
    @echo "Compiling " $<  
    @$(CPP) $(CFLAGS) $<  
builtinParameter1.o: builtinParameter1.cpp builtinParameter1.hpp builtin.hpp symbol.hpp symbolInterface.hpp  
    @echo "Compiling " $<  
    @$(CPP) $(CFLAGS) $<  
mathFunction.o: mathFunction.cpp mathFunction.hpp ../error/error.hpp  
    @echo "Compiling " $<  
    @$(CPP) $(CFLAGS) $<
```

Contenido del ejemplo

13 Funciones matemáticas predefinidas

- Novedades
- Ficheros nuevos y modificados
- Ejercicio

Funciones matemáticas predefinidas

Ejercicio

Ejercicio

- *Poner la funciones predefinidas en castellano: seno, coseno, atan, log, log10, exp, raiz, entero, abs.*

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas**
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 14 Nuevas funciones matemáticas predefinidas
 - Novedades
 - Ficheros nuevos y modificados

Nuevas funciones matemáticas predefinidas

Novedades

Nuevas funciones matemáticas predefinidas

- Función con **cerro** argumentos:
 - **random()**.
- Función con **dos** argumentos:
 - **atan2(x,y)**.
- Nuevos **clases** de la Tabla de Símbolos:
 - Función predefinida con **cerro** argumentos: clase **BuiltinParameter0**.
 - Función predefinida con **dos** argumentos: clase **BuiltinParameter2**.

Nuevas funciones matemáticas predefinidas

Novedades

Función con **cero** argumentos

- **random()**.

Ejemplo

```
print random();  
Print: 0.7159856
```

```
print random();  
Print: 0.65093853
```


Nuevas funciones matemáticas predefinidas

Novedades

Función con **dos** argumentos

- **atan2(x,y).**

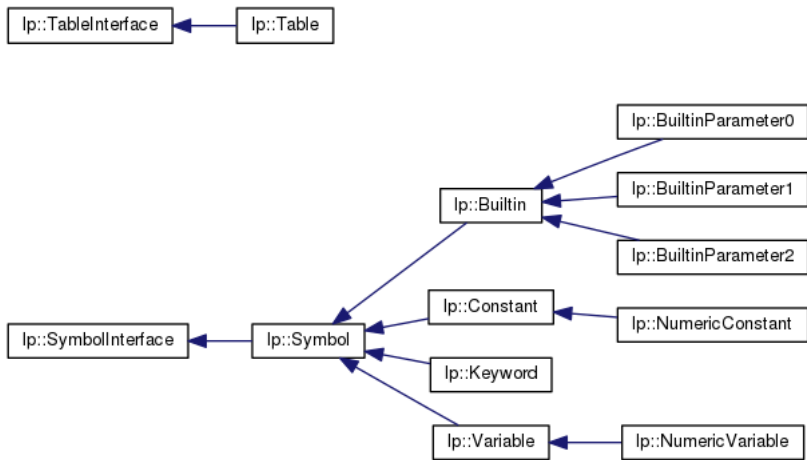
Ejemplo

```
print atan2(1,1);  
Print: 0.7853982
```

```
print atan2(1,2);  
Print: 0.4636476
```

Clases de la Tabla de símbolos (1/2)

- Nuevas clases: **BuiltinParameter0** y **BuiltinParameter2**.



Nuevas funciones matemáticas predefinidas

Novedades

Clases de la Tabla de símbolos (2/2)

- Nuevas clases: **BuiltinParameter0** y **BuiltinParameter2**.

Ip::BuiltinParameter0
- _function
+ BuiltinParameter0()
+ BuiltinParameter0()
+ getFunction()
+ setFunction()
+ operator=()
* function
* BuiltinParameter0()
* BuiltinParameter0()
* getFunction()
* setFunction()
* operator=()

Ip::BuiltinParameter2
- _function
+ BuiltinParameter2()
+ BuiltinParameter2()
+ getFunction()
+ setFunction()
+ operator=()
* function
* BuiltinParameter2()
* BuiltinParameter2()
* getFunction()
* setFunction()
* operator=()

Contenido del ejemplo

- 14 Nuevas funciones matemáticas predefinidas
 - Novedades
 - Ficheros nuevos y modificados

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros nuevos

- **builtinParameter0.hpp**
 - Definición de la clase **BuiltinParameter0**.
- **builtinParameter0.cpp**
 - Código de la clase **BuiltinParameter0**.
- **builtinParameter2.hpp**
 - Definición de la clase **BuiltinParameter2**.
- **builtinParameter2.cpp**
 - Código de la clase **BuiltinParameter2**.

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros modificados (1/2)

- **interpreter.cpp**
 - Inclusión del fichero de cabecera de listas de la STL: **list**.
- **interpreter.l**
 - Inclusión del fichero de cabecera de listas de la STL: **list**.
 - Reconocimiento del símbolo terminal **COMMA**.
- **interpreter.y**
 - Actualización de **YYSTYPE**: **lista de parámetros**.
 - Tipo de nuevos símbolos no terminales: **listOfExp** y **restOfListOfExp**.
 - Definición del token **COMMA**.
 - Reglas gramaticales para el uso de funciones matemáticas con cualquier número de argumentos.

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

Ficheros modificados (2/2)

- **mathFunction.hpp**
 - Prototipo de las nuevas funciones predefinidas.
- **mathFunction.cpp**
 - Código de las nuevas funciones predefinidas.
- **init.hpp**
 - Definición de las nuevas funciones matemáticas predefinidas.
- **init.cpp**
 - Inclusión del fichero de cabecera de listas de la STL.
 - Instalación en la tabla de símbolos de las nuevas funciones matemáticas predefinidas.
- **makefile** del subdirectorio **table**
 - Compilación de nuevos ficheros.

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

interpreter.cpp

- Inclusión del fichero de cabecera de listas de la STL: **list**.

interpreter.cpp

```
...  
#include <list>  
...
```


Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

interpreter.l

- Inclusión del fichero de cabecera de listas de la STL: **list**.
- Reconocimiento del símbolo terminal **COMMA**.

interpreter.l

```
...  
#include <list>  
...  
", "      return  COMMA;
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

interpreter.y

- Actualización de **YYSTYPE**: lista de parámetros.
- Tipo de nuevos símbolos no terminales: `listOfExp` y `restOfListOfExp`.
- Definición del token **COMMA**.

interpreter.y

```
%union {  
    double number;  
    char * identifier;  
    std::list<double> * parameters;  
}  
...  
%type <parameters> listOfExp restOfListOfExp  
...  
%nonassoc COMMA
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

interpreter.y

- **Reglas gramaticales** para el uso de funciones matemáticas con cualquier número de argumentos.

interpreter.y

```
exp: ...
    | BUILTIN LPAREN listOfExp RPAREN
    ;

listOfExp: /* Empty list of numeric expressions */
    | exp restOfListOfExp
    ;

restOfListOfExp: /* Empty list of numeric expressions */
    | COMMA exp restOfListOfExp
    ;
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

init.hpp

- Definición de las **nuevas funciones matemáticas predefinidas**.

init.hpp

```
...
static struct {
    std::string name;
    lp::TypePointerDoubleFunction_0 function;
} function_0 [] = { {"random", Random},
                  {"", 0}
                };

static struct {
    std::string name;
    lp::TypePointerDoubleFunction_2 function;
} function_2 [] = { {"atan2", Atan2},
                  {"", 0}
                };
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

builtinParameter0.hpp

- New type definition: **TypePointerDoubleFunction_0**

builtinParameter0.hpp

```
typedef double (*TypePointerDoubleFunction_0)();
```

builtinParameter2.hpp

- New type definition: **TypePointerDoubleFunction_2**

builtinParameter2.hpp

```
typedef double (*TypePointerDoubleFunction_2)(double x,double y);
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

init.cpp

- Inclusión del fichero de cabecera de listas de la STL: **list**.
- **Instalación** en la tabla de símbolos de las **nuevas funciones matemáticas predefinidas**.

init.cpp

```
#include<list>
...
lp::BuiltinParameter0 *f0;
for (i=0; function_0[i].name.compare("")!=0; i++) {
    f0 = new lp::BuiltinParameter0(function_0[i].name, BUILTIN, 0, function_0[i].function);
    t.installSymbol(f0);
}

lp::BuiltinParameter2 *f2;
for (i=0; function_2[i].name.compare("")!=0; i++) {
    f2 = new lp::BuiltinParameter2(function_2[i].name, BUILTIN, 2, function_2[i].function);
    t.installSymbol(f2);
}
```

Nuevas funciones matemáticas predefinidas

Ficheros nuevos y modificados

makefile del subdirectorio table

- Compilación de los nuevos ficheros
 - builtinParameter0.hpp, builtinParameter0.cpp.
 - builtinParameter2.hpp, builtinParameter2.cpp.

makefile del subdirectorio table

```
OBJECTS= table.o symbol.o variable.o numericVariable.o \  
init.o constant.o numericConstant.o keyword.o \  
builtin.o builtinParameter1.o mathFunction.o builtinParameter0.o builtinParameter2.o  
...  
builtinParameter0.o: builtinParameter0.cpp builtinParameter0.hpp builtin.hpp \  
symbol.hpp symbolInterface.hpp  
@echo "Compiling " $<  
@$(CPP) $(CFLAGS) $<  
  
builtinParameter2.o: builtinParameter2.cpp builtinParameter2.hpp builtin.hpp \  
symbol.hpp symbolInterface.hpp  
@echo "Compiling " $<  
@$(CPP) $(CFLAGS) $<
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento-reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST**
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 15 Generación de código intermedio AST
 - Novedades
 - Ficheros nuevos y modificados

Generación de código intermedio AST

Novedades

Generación de código intermedio AST

- Uso de los **árboles de sintaxis abstracta (AST)** para evaluar las expresiones.
- **Nuevas clases** para la generación de código AST.
- Se controla el modo de ejecución del intérprete: variable **interactiveMode**.

Generación de código intermedio AST

Novedades

Generación de código intermedio

- Uso de los **árboles de sintaxis abstracta (AST)** para evaluar las expresiones.

Ejemplo: asignación

```
dato = 2 * PI;
assignment_node: =
  dato
  MultiplicationNode: *
  NumberNode: 2
  ConstantNode: PI (Type: 263)
```

Ejemplo: escribir

```
print dato;
PrintStmt: print
  VariableNode: dato (Type: 263)

Print: 6.283185
```

Generación de código intermedio AST

Novedades

Generación de código intermedio

- Uso de los **árboles de sintaxis abstracta (AST)** para evaluar las expresiones.

Ejemplo: asignación

```
dato = 2 * sin(PI/4);
assignment_node: =
  dato
  MultiplicationNode: *
  NumberNode: 2
  BuiltinFunctionNode_1:
  sin
  DivisionNode: /
  ConstantNode: PI (Type: 263)
  NumberNode: 4
```

Ejemplo: escribir

```
print dato;
PrintStat: print
  VariableNode: dato (Type: 263)

Print: 1.414214
```

Generación de código intermedio AST

Novedades

Generación de código intermedio

- Uso de los **árboles de sintaxis abstracta (AST)** para evaluar las expresiones.

Ejemplo: escribir

```
print sin(PI/2);  
PrintStmt: print  
  BuiltinFunctionNode_1:  
    sin  
  DivisionNode: /  
  ConstantNode: PI (Type: 263)  
  NumberNode: 2
```

```
Print: 1
```

Generación de código intermedio AST

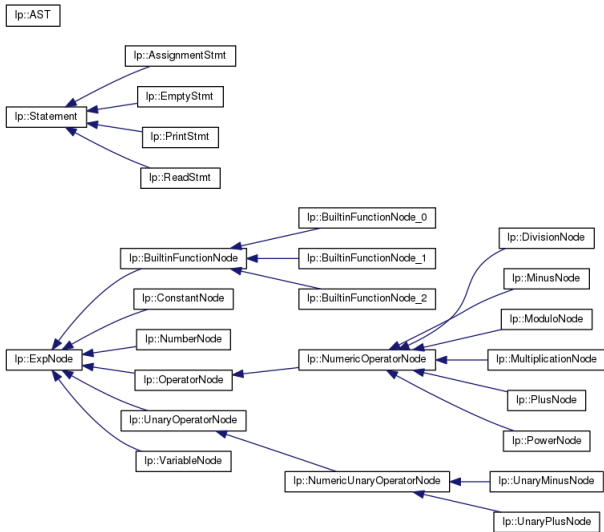
Novedades

Nuevas clases para la generación de código intermedio AST

- **AST**
- **Statement** y clases derivadas
- **ExpNode** y clases derivadas

Nuevas clases para la generación de código intermedio AST

- Jerarquía de las clases.



Nuevas clases para la generación de código intermedio AST

- **AST**

Ip::AST
- stmts
+ AST() + print() + evaluate()

Generación de código intermedio AST

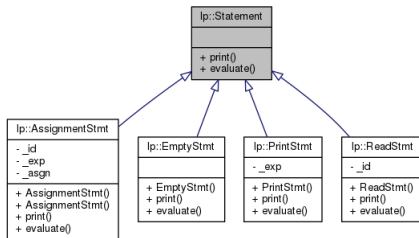
Novedades

Nuevas clases para la generación de código intermedio AST

- **Statement** y clases derivadas.
 - **AssigmentStmt**
 - **EmptyStmt**
 - **PrintStmt**
 - **ReadStmt**

Nuevas clases para la generación de código intermedio AST

- **Statement** y clases derivadas.



Generación de código intermedio AST

Novedades

Nuevas clases para la generación de código intermedio AST

- **ExpNode** y clases derivadas
 - **NumberNode**.
 - **NumericConstantNode**.
 - **NumericVariableNode**.
 - **OperatorNode** y clases derivadas.
 - **UnaryOperatorNode** y clases derivadas.
 - **BuiltinFunctionNode** y clases derivadas.

Generación de código intermedio AST

Novedades

Nuevas clases para la generación de código intermedio AST

- **OperatorNode** y clases derivadas.
 - **NumericOperatorNode**.
 - **PlusNode**.
 - **MinusNode**.
 - **MultiplicationNode**.
 - **DivisionNode**.
 - **ModuloNode**.
 - **PowerNode**.

Generación de código intermedio AST

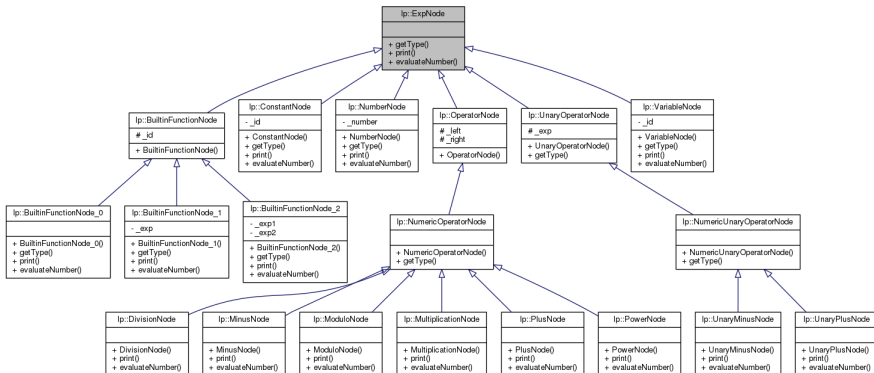
Novedades

Nuevas clases para la generación de código intermedio AST

- **UnaryOperatorNode** y clases derivadas.
 - **NumericUnaryOperatorNode.**
 - **UnaryMinusNode.**
 - **UnaryPlusNode.**

Nuevas clases para la generación de código intermedio AST

- **ExpNode** y clases derivadas.



Contenido del ejemplo

- 15 Generación de código intermedio AST
 - Novedades
 - Ficheros nuevos y modificados

Generación de código intermedio AST

Ficheros nuevos y modificados

Ficheros nuevos

- **ast.hpp**
 - Definición de la clase **AST**.
- **ast.cpp**
 - Código de la clase **AST**.

Generación de código intermedio AST

Ficheros nuevos y modificados

Ficheros modificados (1/2)

- **interpreter.cpp**
 - Inclusión de **ast.hpp**.
 - Declaración de la raíz del árbol de sintaxis abstracta.
 - Declaración de la variable de control de ejecución: **interactiveMode**.
 - **Impresión y evaluación** del árbol de sintaxis abstracta.
- **interpreter.l**
 - Inclusión de **ast.hpp**.
- **interpreter.y**
 - Inclusión de **ast.hpp**.
 - Modificación de **YYSTYPE**.
 - Referencia a la declaración de la variable de control de ejecución: **interactiveMode**.
 - Uso de la variable de control de ejecución: **interactiveMode**.

Generación de código intermedio AST

Ficheros nuevos y modificados

Ficheros modificados (2/2)

- **init.hpp**
 - Inclusión de **ast.hpp**.
- **Makefile**
 - Acceso al makefile del subdirectorio **ast**.
- **Makefile** del subdirectorio **ast**.
 - Compilación de los nuevos ficheros: **ast.hpp**, **ast.cpp**.
- **Doxyfile**
 - Generación de la documentación del subdirectorio **ast**.

Generación de código intermedio AST

Ficheros nuevos y modificados

interpreter.cpp

- Inclusión de **ast.hpp**.
- Declaración de la **raíz** del árbol de sintaxis abstracta.
- Declaración de la variable de **control** de ejecución: **interactiveMode**.
- **Impresión y evaluación** del árbol de sintaxis abstracta.

interpreter.cpp

```
...
#include "ast/ast.hpp"
...
lp::AST *root; //!< Root of the abstract syntax tree AST
...
bool interactiveMode; //!< Control the interactive mode of execution of the interpreter
...
root->print();
root->evaluate();
```

Generación de código intermedio AST

Ficheros nuevos y modificados

interpreter.l

- Inclusión de **ast.hpp**.

interpreter.l

```
...  
#include "../ast/ast.hpp"  
...
```

Generación de código intermedio AST

Ficheros nuevos y modificados

interpreter.y

- Inclusión de **ast.hpp**.
- Modificación de **YYSTYPE**.

interpreter.y

```
#include "../ast/ast.hpp"
...
%union {
    char * identifier;
    double number;

    /* New in example 15 */
    lp::ExpNode *expNode;
    std::list<lp::ExpNode *> *parameters;
    std::list<lp::Statement *> *stmts;
    lp::Statement *st;
    lp::AST *prog;
}
```

Generación de código intermedio AST

Ficheros nuevos y modificados

interpreter.y

- Referencia a la declaración de la variable de control de ejecución: **interactiveMode..**
- Uso de la variable de control de ejecución: **interactiveMode..**

interpreter.y

```
extern bool interactiveMode; //!< Control the interactive mode of execution of the interpreter
...
stmtlist: /* empty: epsilon rule */
  | stmtlist stmt
  { $$ = $1;
    $$->push_back($2);
    if (interactiveMode == true) {
      $2->print();
      $2->evaluate();
    }
  }
  ...
```

Generación de código intermedio AST

Ficheros nuevos y modificados

init.hpp

- Inclusión de **ast.hpp**.

init.hpp

```
...  
#include "../ast/ast.hpp"  
...
```

Generación de código intermedio AST

Ficheros nuevos y modificados

Makefile principal

- Acceso al **makefile** del subdirectorio **ast**.

Makefile principal

```
...
OBJECTS-AST = ast/*.o # New in example 15
...
INCLUDES = ...
    ./ast/ast.hpp
...
$(NAME).exe : parser-dir error-dir table-dir ast-dir $(OBJECTS)
    @echo "Generating " $(NAME).exe
    @$$(CPP) $(OBJECTS) $(OBJECTS-PARSER) $(OBJECTS-ERROR) $(OBJECTS-TABLE) $$(OBJECTS-AST)
...
ast-dir:
    @echo "Accessing directory ast"
    @echo
    @make -C ast/
    @echo
```


Generación de código intermedio AST

Ficheros nuevos y modificados

Makefile del subdirectorio ast

- Compilación de los nuevos ficheros: **ast.hpp**, **ast.cpp**.

Makefile del subdirectorio ast

```
NAME=ast
...
# Objects
OBJECTS= $(NAME).o
...
all: $(OBJECTS)

$(NAME).o: $(NAME).cpp $(INCLUDES)
    @echo "Compiling " $<
    @$(CPP) $(CFLAGS) $<
    @echo
```

Generación de código intermedio AST

Ficheros nuevos y modificados

Ficheros modificados (10/10)

- **Doxyfile**
 - Generación de la documentación del subdirectorio **ast**.

Makefile

```
INPUT = interpreter.cpp parser error includes table ast
```

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos**
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

16 Constantes y variables lógicas, operadores relacionales y lógicos

- Novedades
- Ficheros nuevos y modificados
- Ejercicio

Constantes y variables lógicas, operadores relacionales y lógicos

Novedades

Novedades

- Constantes y variables **lógicas**.
- Operadores **relacionales** y **lógicos**.
- **Conversión** dinámica del tipo de variable.
- **Nuevas clases** para la generación de código AST.

Constantes y variables lógicas, operadores relacionales y lógicos

Novedades

Constantes y variables lógicas

- **Constantes** lógicas: **true**, **false**.
- **Variables** lógicas.

Ejemplo: true

```
dato = true;  
print dato;  
Print: true
```

Ejemplo: false

```
dato = false;  
print dato;  
Print: false
```

Constantes y variables lógicas, operadores relacionales y lógicos

Novedades

Conversión dinámica del tipo de variable.

- Las **variables** pueden **cambiar de tipo** numérico a lógico y viceversa.

Ejemplo

```
dato = 2;  
print dato;  
Print: 2
```

```
dato = true;  
print dato;  
Print: true
```

```
dato = 2;  
print dato;  
Print: 2
```

Constantes y variables lógicas, operadores relacionales y lógicos

Novedades

Constantes y variables lógicas

- **Generación de código AST:** operadores lógicos y relacionales.

Ejemplo AST: asignación

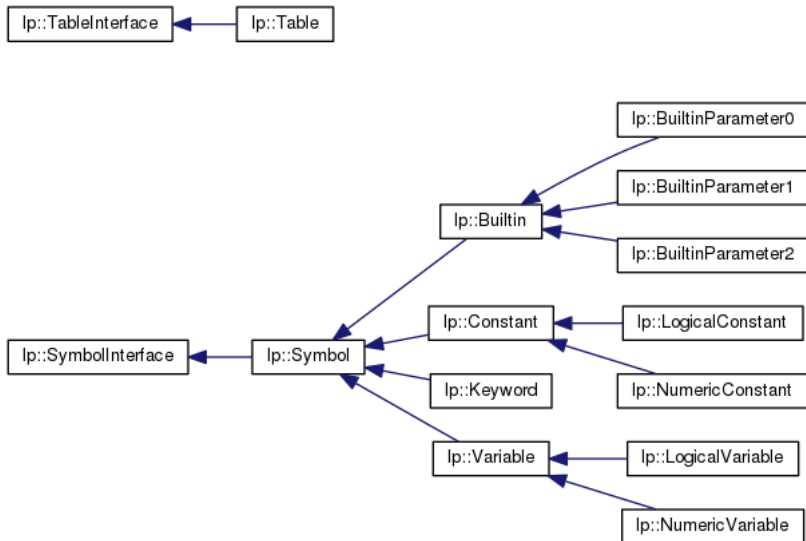
```
dato = 2;  
assignment_node: =  
    dato  
    NumberNode: 2
```

Ejemplo AST: operadores

```
print (dato > 0) && (dato < 10);  
PrintStmt: print  
    AndNode: &&  
    GreaterThanNode: >  
    VariableNode: dato (Type: 263)  
    NumberNode: 0  
    LessThanNode: <  
    VariableNode: dato (Type: 263)  
    NumberNode: 10  
  
Print: true
```

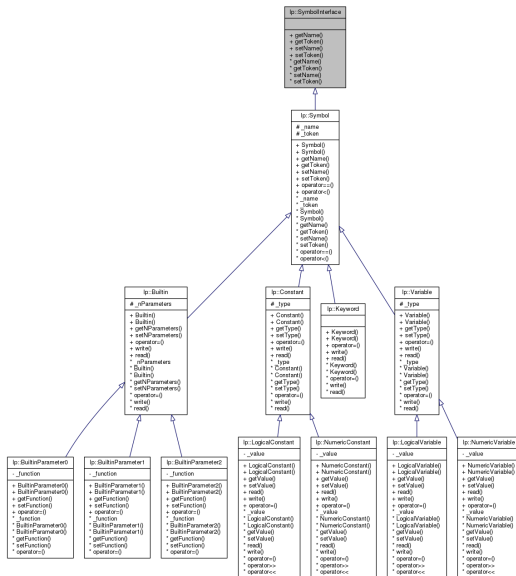

Clases de la Tabla de símbolos

- Nuevas clases: **logicalConstant** y **logicalVariable**.



Clases de la Tabla de símbolos

- Nuevas clases: **logicalConstant** y **logicalVariable**.



Constantes y variables lógicas, operadores relacionales y lógicos

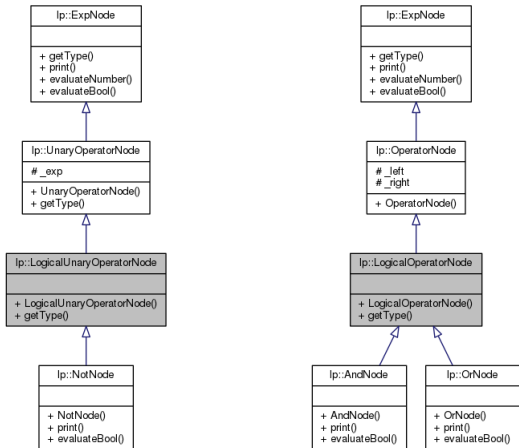
Novedades

Nuevas clases para la generación de código AST

- **LogicalUnaryOperatorNode** y **NotNode**.
- **LogicalOperatorNode**, **AndNode** y **OrNode**.
- **RelationalOperatorNode** y clases derivadas.

Nuevas clases para la generación de código AST

- **LogicalUnaryOperatorNode** y **NotNode**.
- **LogicalOperatorNode**, **AndNode** y **OrNode**.



Constantes y variables lógicas, operadores relacionales y lógicos

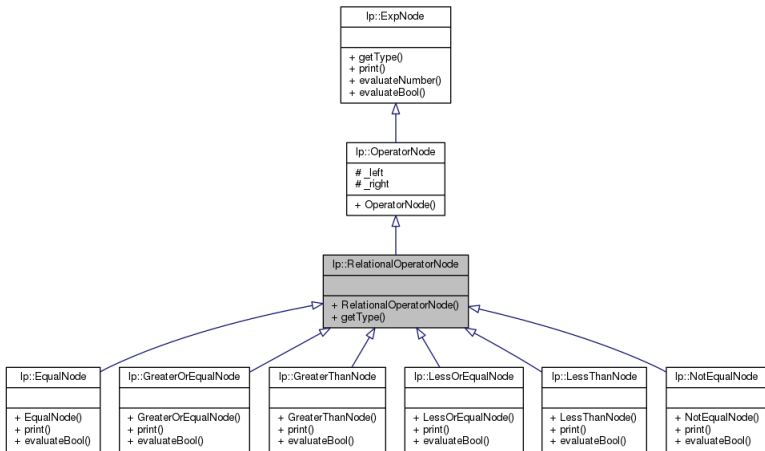
Novedades

Nuevas clases para la generación de código AST

- **RelationalOperatorNode** y clases derivadas.
 - **EqualNode**
 - **GreaterOrEqualNode**
 - **GreaterThanNode**
 - **LessOrEqualNode**
 - **LessThanNode**
 - **NotEqualNode**

Nuevas clases para la generación de código AST

- **RelationalOperatorNode** y clases derivadas



Contenido del ejemplo

- 16 Constantes y variables lógicas, operadores relacionales y lógicos
 - Novedades
 - Ficheros nuevos y modificados
 - Ejercicio

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

Ficheros nuevos

- **logicalConstant.hpp**
 - Definición de la clase **LogicalConstant**.
- **logicalConstant.cpp**
 - Código de la clase **LogicalConstant**
- **logicalVariable.hpp**
 - Definición de la clase **logicalVariable**.
- **logicalVariable.cpp**
 - Código de la clase **logicalVariable**

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

Ficheros modificados (1/2)

- **interpreter.l**
 - Reconocimiento de los **operadores relacionales y lógicos**.
- **interpreter.y**
 - **Definición** de los componentes léxicos de los **operadores relacionales o lógicos**.
 - **Reglas sintácticas** para reconocer las sentencias con **operadores relacionales o lógicos**.

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

Ficheros modificados (2/2)

- **ast.hpp**
 - **Definición** de las **nuevas clases** de generación de código AST.
- **ast.cpp**
 - **Código** de las **nuevas clases** de generación de código AST.
- **Makefile** del subdirectorio **table**.
 - Compilación de los nuevos ficheros:
 - **logicalConstant.hpp**, **logicalConstant.cpp**.
 - **logicalVariable.hpp**, **logicalVariable.cpp**.

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

interpreter.l

- Reconocimiento de los **operadores relacionales y lógicos**.

interpreter.l

```
"==" { return EQUAL; }
"!=" { return NOT_EQUAL; }
">=" { return GREATER_OR_EQUAL; }
"<=" { return LESS_OR_EQUAL; }
">" { return GREATER_THAN; }
"<" { return LESS_THAN; }

"!" { return NOT; }
"||" { return OR; }
"&&" { return AND; }
```

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

interpreter.y (1/2)

- **Definición** de los componentes léxicos de los **operadores relacionales o lógicos**.

interpreter.y

```
...
%left OR
%left AND

%nonassoc GREATER_OR_EQUAL LESS_OR_EQUAL GREATER_THAN LESS_THAN EQUAL NOT_EQUAL

%left NOT
...
```

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

interpreter.y (2/2)

- Reglas sintácticas para reconocer las sentencias con operadores relacionales o lógicos.

interpreter.y

```
exp: ...
| exp GREATER_THAN exp { ... }
| exp GREATER_OR_EQUAL exp { ... }
| exp LESS_THAN exp { ... }
| exp LESS_OR_EQUAL exp { ... }
| exp EQUAL exp { ... }
| exp NOT_EQUAL exp { ... }
| exp AND exp { ... }
| exp OR exp { ... }
| NOT exp { ... }
;
```

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

ast.hpp

- **Definición** de las nuevas clases de generación de código AST.
 - Clases de los **operadores relacionales**
 - RelationalOperatorNode y clases derivadas: EqualNode, NotEqualNode. GreaterOrEqualNode, GreaterThanNode. LessOrEqualNode, LessThanNode.
 - Clases de los **operadores lógicos**
 - LogicalUnaryOperatorNode y NotNode.
 - LogicalOperatorNode, AndNode y OrNode.

Constantes y variables lógicas, operadores relacionales y lógicos

Ficheros nuevos y modificados

ast.cpp

- **Código** de las nuevas clases de generación de código AST.
 - Clases de los **operadores relacionales**
 - RelationalOperatorNode y clases derivadas: EqualNode, NotEqualNode. GreaterOrEqualNode, GreaterThanNode. LessOrEqualNode, LessThanNode.
 - Clases de los **operadores lógicos**
 - LogicalUnaryOperatorNode y NotNode.
 - LogicalOperatorNode, AndNode y OrNode.

Contenido del ejemplo

- 16 Constantes y variables lógicas, operadores relacionales y lógicos
 - Novedades
 - Ficheros nuevos y modificados
 - Ejercicio

Constantes y variables lógicas, operadores relacionales y lógicos

Ejercicio

Ejercicio

- *Cambiar las constantes lógicas **true** y **false** por **verdadero** y **falso**.*

Ejemplos

- 1 Reconocimiento de expresiones aritméticas simples
- 2 Análisis de un fichero
- 3 Reconocimiento de operadores unarios
- 4 Evaluación de expresiones aritméticas
- 5 Separador de expresiones y nuevos operadores
- 6 Recuperación de errores de ejecución
- 7 Variables y tabla de símbolos
- 8 Conflicto de desplazamiento-reducción
- 9 Resolución del conflicto de desplazamiento - reducción y sentencias de lectura y escritura
- 10 Constantes predefinidas que se pueden modificar
- 11 Constantes predefinidas que no se pueden modificar
- 12 Palabras claves pre-instaladas en la tabla de símbolos
- 13 Funciones matemáticas predefinidas
- 14 Nuevas funciones matemáticas predefinidas
- 15 Generación de código intermedio AST
- 16 Constantes y variables lógicas, operadores relacionales y lógicos
- 17 Sentencias de control de flujo y conflicto del "else danzante"

Contenido del ejemplo

- 17 Sentencias de control de flujo y conflicto del “else danzante”
 - Novedades
 - Ficheros modificados

Sentencias de control de flujo y conflicto del “else danzante”

Novedades

Novedades

- Sentencias de control de flujo: **if**, **while**.
- **Conflicto de desplazamiento – reducción** provocado por la sentencia del “**else danzante**”.
- **Nuevas clases** para la generación de código AST.

Sentencias de control de flujo y conflicto del “else danzante”

Novedades

Sentencias de control de flujo (1/2)

- Sentencia **if**.

Ejemplo: código AST

```
if (a>0) b = a; else b = -a;
IfStmt:
  GreaterThanNode: >
  VariableNode: a (Type: 268)
  NumberNode: 0
  assignment_node: =
  b
  VariableNode: a (Type: 268)

  assignment_node: =
  b
  UnaryMinusNode: -
  VariableNode: a (Type: 268)
```

Sentencias de control de flujo y conflicto del “else danzante”

Novedades

Sentencias de control de flujo (2/2)

- Sentencia **while**.

Ejemplo: código AST

```
while (n>0) { print n; n = n-1;}
WhileStmt:
    GreaterThanNode: >
    VariableNode: n (Type: 268)
    NumberNode: 0

    BlockStmt:
    PrintStmt: print
    VariableNode: n (Type: 268)

    assignment_node: =
    n
    MinusNode: -
    VariableNode: n (Type: 268)
    NumberNode: 1
    ...
```

Sentencias de control de flujo y conflicto del “else danzante”

Novedades

Conflicto de desplazamiento – reducción

- Provocado por la sentencia del “**else danzante**”.

Comprobación del conflicto (1/2)

```
bison -v interpreter.y  
interpreter.y: aviso: 1 conflicto desplazamiento/reducción [-Wconflicts-sr]
```

Sentencias de control de flujo y conflicto del “else danzante”

Novedades

Conflicto de desplazamiento – reducción

- Provocado por la sentencia del “else danzante”.

Comprobación del conflicto (2/2): fichero interpreter.output

```
...
Estado 89 conflictos: 1 desplazamiento/reducción
...
Estado 89
14 if: IF controlSymbol cond stmt .
15 | IF controlSymbol cond stmt . ELSE stmt

ELSE desplazar e ir al estado 95

ELSE [reduce usando la regla 14 (if)]
$default reduce usando la regla 14 (if)
```


Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** **ε** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ **ε** **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **∈** cond **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ stmtlist **IF** **∈** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ **∈** **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER** RPAREN VARIABLE ASSIGNMENT **NUMBER** SEMICOLON **ELSE** VARIABLE ASSIGNMENT **NUMBER** SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL **NUMBER** RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **stmtlist IF ε** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **ε IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** **ε** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **ε** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol ε cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ε **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **∈** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **VARIABLE ASSIGNMENT NUMBER SEMICOLON** ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** **∈** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **∈** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **∈** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** **∈** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **∈** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol **∈** cond **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN **VARIABLE** EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** **∈** LPAREN VARIABLE EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **∈** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL **NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN **VARIABLE EQUAL NUMBER RPAREN** VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL **NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol LPAREN **VARIABLE EQUAL NUMBER RPAREN** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ stmtlist **IF** **ε** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **ε** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT **NUMBER SEMICOLON ELSE** VARIABLE ASSIGNMENT **NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** **ε** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **ε** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **ε** cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** **ε** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **ε** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ **stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ **stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if
⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT
NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE
VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE
VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER
SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT
NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT
NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

program

⇒ stmtlist

⇒ stmtlist stmt

⇒ stmtlist if

⇒ stmtlist IF controlSymbol cond stmt

(Se asocia ELSE al IF más cercano)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

program

⇒ stmtlist

⇒ stmtlist stmt

⇒ stmtlist **if**

⇒ stmtlist **IF** controlSymbol cond stmt

(Se asocia **ELSE** al **IF** más cercano)

⇒ stmtlist **IF** controlSymbol cond IF controlSymbol cond stmt **ELSE** stmt

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond stmt **ELSE** asgn SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT exp SEMICOLON (*)

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **LPAREN** exp **RPAREN** VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **LPAREN** exp **EQUAL** exp **RPAREN** VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **LPAREN** exp **EQUAL** exp **RPAREN** VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

program

⇒ stmtlist

⇒ stmtlist stmt

⇒ stmtlist if

⇒ stmtlist IF controlSymbol cond stmt

(Se asocia ELSE al IF más cercano)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

program

- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if
- ⇒ stmtlist IF controlSymbol cond stmt (Se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Primera derivación por la derecha en orden inverso (2/2)

program

⇒ stmtlist

⇒ stmtlist stmt

⇒ stmtlist if

⇒ stmtlist IF controlSymbol cond stmt

(Se asocia ELSE al IF más cercano)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE stmt

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE asgn SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON (*)

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ **IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol **cond** **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL exp RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN exp EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ IF LPAREN VARIABLE EQUAL NUMBER RPAREN IF LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (1/2)

- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL exp RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN exp EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN **IF** LPAREN VARIABLE EQUAL NUMBER RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```


Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

```
program
⇒ stmtlist
⇒ stmtlist stmt
⇒ stmtlist if (No se asocia ELSE al IF más cercano)
⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
⇒ ...
```

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if (No se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ...

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if (No se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ...

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if (No se asocia ELSE al IF más cercano)
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** stmt
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** asgn SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON** (*)
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond **stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond **asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond **VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **cond** **VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ ...

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist **if** (No se asocia **ELSE** al **IF** más cercano)
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** stmt
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** asgn SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON** (*)
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond **stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON**
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol **cond** VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ...

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if (No se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ...

Conflicto de desplazamiento - reducción

- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

program

⇒ stmtlist

⇒ stmtlist stmt

⇒ stmtlist if

(No se asocia **ELSE** al **IF** más cercano)

⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** stmt

⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** asgn SEMICOLON

⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT exp SEMICOLON

⇒ stmtlist **IF** controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond if **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

(*)

⇒ stmtlist **IF** controlSymbol cond IF controlSymbol cond stmt **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond asgn SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ stmtlist **IF** controlSymbol cond **IF** controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON **ELSE** VARIABLE ASSIGNMENT NUMBER SEMICOLON

⇒ ...

Conflicto de desplazamiento - reducción

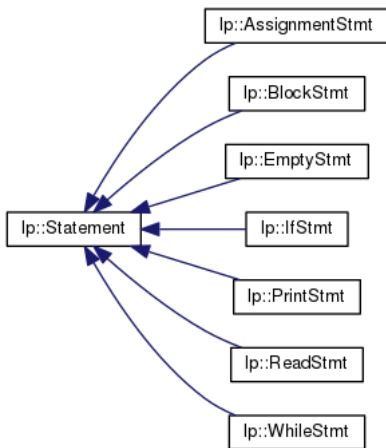
- La sentencia **if** se puede generar de **dos** maneras.

Segunda derivación por la derecha en orden inverso (2/2)

- program
- ⇒ stmtlist
- ⇒ stmtlist stmt
- ⇒ stmtlist if (No se asocia ELSE al IF más cercano)
- ⇒ stmtlist IF controlSymbol cond stmt ELSE stmt
- ⇒ stmtlist IF controlSymbol cond stmt ELSE asgn SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT exp SEMICOLON
- ⇒ stmtlist IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond if ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON (*)
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond stmt ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond asgn SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT exp SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol cond VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ stmtlist IF controlSymbol cond IF controlSymbol LPAREN exp RPAREN VARIABLE ASSIGNMENT NUMBER SEMICOLON ELSE VARIABLE ASSIGNMENT NUMBER SEMICOLON
- ⇒ ...

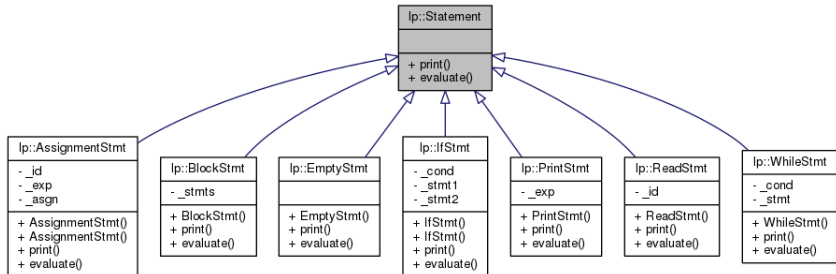
Nuevas clases para la generación de código AST.

- **BlockStmt**, **IfStmt** y **WhileStmt**.



Nuevas clases para la generación de código AST.

- **BlockStmt**, **IfStmt** y **WhileStmt**.



Contenido del ejemplo

- 17 Sentencias de control de flujo y conflicto del “else danzante”
 - Novedades
 - Ficheros modificados

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

Ficheros modificados (1/2)

- **interpreter.l**

- Reconocimiento de los delimitadores de sentencias de bloque: “{” y “}”:

- **interpreter.y**

- Definición de nuevos símbolos **terminales**:
LETFCURLYBRACKET, RIGHTCURLYBRACKET, IF, ELSE, WHILE.
- Tipo de dato de nuevos símbolos **no** terminales:
 - **cond, block, if, while.**
- Reglas sintácticas para reconocer las **sentencias de control**:
 - **block, cond, if, while** y **controlSymbol.**

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

Ficheros modificados (2/2)

- **init.hpp**
 - Definición de nuevas **palabras reservadas**:
if, else, while.
- **ast.hpp**
 - Definición de las **nuevas clases** de generación de código AST:
IfStmt, WhileStmt y BlockStmt.
- **ast.cpp**
 - Código de las **nuevas clases** de generación de código AST:
IfStmt, WhileStmt y BlockStmt.

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.l

- Reconocimiento de los delimitadores de sentencias de bloque:
“{” y “}”:

interpreter.l

```
"{" { return LEFTCURLYBRACKET; }
```

```
"}" { return RIGHTCURLYBRACKET; }
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.y (1/5)

- Definición de nuevos símbolos **terminales**:

**LETFCURLYBRACKET, RIGHTCURLYBRACKET,
IF, ELSE, WHILE.**

interpreter.y

```
...  
%token PRINT READ IF ELSE WHILE  
  
%token LETFCURLYBRACKET RIGHTCURLYBRACKET  
...
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.y (2/5)

- Tipo de dato de nuevos símbolos no terminales:
 - `cond`, `block`, `if`, `while`.

interpreter.y

```
...
%type <expNode> exp cond
...
%type <st> stmt asgn print read if while block
...
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.y (3/5)

- **controlSymbol**
 - Permite controlar si la ejecución es **interactiva o no** dentro de una sentencia **if** o **while**.

interpreter.y

```
stmtlist: ...
  | stmtlist stmt
  { ...
    // Control the interactive mode of execution of the interpreter
    if (interactiveMode == true && control == 0)
      { ... }
  }
;

...
controlSymbol: /* Epsilon rule */ { control++; }
;
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.y (4/5)

- Reglas sintácticas para reconocer las **sentencias de control** (1/2):
 - **block**, **cond**, **if**, **while** y **controlSymbol**.

interpreter.y

```
stmt: ...
    | if { ... }
    | while { ... }
    | block { ... }
    ;

...
block: LETFCURLYBRACKET stmtlist RIGHTCURLYBRACKET { ... }
    ;

controlSymbol: /* Epsilon rule */ { control++; }
    ;
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

interpreter.y (5/5)

- Reglas sintácticas para reconocer las **sentencias de control** (2/2):
 - **block**, **cond**, **if**, **while** y **controlSymbol**.

interpreter.y

```
if: IF controlSymbol cond stmt { ... control--; }
  | IF controlSymbol cond stmt ELSE stmt { ... control--; }
  ;

while: WHILE controlSymbol cond stmt { ... control--; }
      ;

cond: LPAREN exp RPAREN
     ;
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

init.hpp

- Definición de nuevas **palabras reservadas** con sus componentes léxicos: **if**, **else**, **while**.

init.hpp

```
static struct {
    std::string name ;
    int token;
} keyword[] = {
    {"print",  PRINT},
    {"read",   READ},
    {"if",     IF},
    {"else",   ELSE},
    {"while",  WHILE},
    {"",       0}
};
```

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

init.hpp

- Definición de nuevas **palabras reservadas** con sus componentes léxicos: **if**, **else**, **while**.

init.hpp

```
static struct {
    std::string name ;
    int token;
} keyword[] = {
    {"print",  PRINT},
    {"read",   READ},
    {"if",     IF},
    {"else",   ELSE},
    {"while",  WHILE},
    {"",       0}
};
```


Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

ast.hpp

- **Definición** de las **nuevas clases** de generación de código AST.
 - **IfStmt**,
 - **WhileStmt**
 - **BlockStmt**

Sentencias de control de flujo y conflicto del “else danzante”

Ficheros modificados

ast.cpp

- **Código** de las **nuevas clases** de generación de código AST.
 - **IfStmt**,
 - **WhileStmt**
 - **BlockStmt**

PROCESADORES DE LENGUAJES

Bison y flex

Prof. Dr. Nicolás Luis Fernández García
Departamento de Informática y Análisis Numérico

Grado de Ingeniería Informática
Especialidad de Computación
Tercer curso. Segundo cuatrimestre
Escuela Politécnica Superior de Córdoba
Universidad de Córdoba