Compiladores
Laboratorio 5
Parser

Objetivo

Evaluar expresiones aritméticas simples ingresadas como texto, construyendo y utilizando un árbol de sintaxis abstracta (AST).

Clases

- **Clase Token**
  Responsable de representar los elementos léxicos (tokens) de la expresión, como números, operadores y paréntesis. Cada objeto Token almacena el tipo de símbolo y su representación textual.

- **Clase Scanner**
  Se encarga del análisis léxico, es decir, de recorrer la cadena de entrada carácter por carácter y segmentarla en una secuencia de tokens válidos, que serán utilizados en la etapa de análisis sintáctico. El Scanner identifica los siguientes tipos de tokens:

  - Números (NUM)
  - Operadores aritméticos: suma (+), resta (-), multiplicación (*) y división (/)
  - Paréntesis: apertura (() y cierre ())
  - Fin de expresión (END)
  - Caracteres no reconocidos (ERR), que representan errores léxicos

- **Clase Parser**
  Se encarga del análisis sintáctico. A partir de la secuencia de tokens generada por el Scanner, construye un árbol de sintaxis abstracta (AST) que refleja la estructura jerárquica de la expresión aritmética. Este proceso se basa en una gramática recursiva descendente, definida de la siguiente manera:

  Exp → Term (("+" | "-") Term)*
  Term → Factor (("*" | "/") Factor)*
  Factor → Num | "(" Exp ")"

  Esta gramática permite reconocer expresiones aritméticas con operaciones de suma, resta, multiplicación, división y el uso de paréntesis para agrupar subexpresiones, respetando la precedencia y asociatividad de los operadores.

- **Clases Exp, BinaryExp y NumberExp**
  Estas clases representan los nodos del Árbol de Sintaxis Abstracta (AST) utilizado para modelar expresiones aritméticas:

  - Exp: Clase base abstracta que define la interfaz común para todos los tipos de expresiones.

  - BinaryExp: Representa operaciones binarias como suma, resta, multiplicación y división. Contiene referencias a las subexpresiones izquierda y derecha, así como el operador correspondiente.

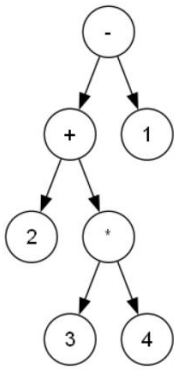
  - NumberExp: Representa un valor numérico entero dentro de la expresión.

  La clase abstracta Exp define dos métodos virtuales puros fundamentales:

  - void print(): Imprime la expresión en notación infija, permitiendo visualizar la estructura del árbol sintáctico de manera legible.

  - int eval(): Evalúa recursivamente la expresión y retorna su resultado numérico.

Ejemplo
Al ingresar la expresión 2 + 3 * 4 - 1, el programa construye el siguiente árbol de sintaxis abstracta (AST), respetando la precedencia de operadores:



Como resultado, el programa imprime:

expr: 2 + 3 * 4 - 1
eval: 11

Ejercicio
Extender la calculadora a un lenguaje imperativo, para esto debe modificar la gramática:

```
Program ::= StmtList
StmtList ::= Stmt (';' Stmt)*
Stmt ::= id '=' Exp | 'print' '(' Exp ')'
Exp ::= Term (('+' | '-') Term)*
Term ::= Factor (('*' | '/') Factor)*
Factor ::= id | Num | '(' Exp ')'
```

Esta extensión permite:

- Declarar variables y asignarles valores mediante expresiones.
- Imprimir valores de expresiones mediante la sentencia print.
- Soportar secuencias de sentencias mediante el uso de punto y coma para separar las instrucciones.

Ayuda

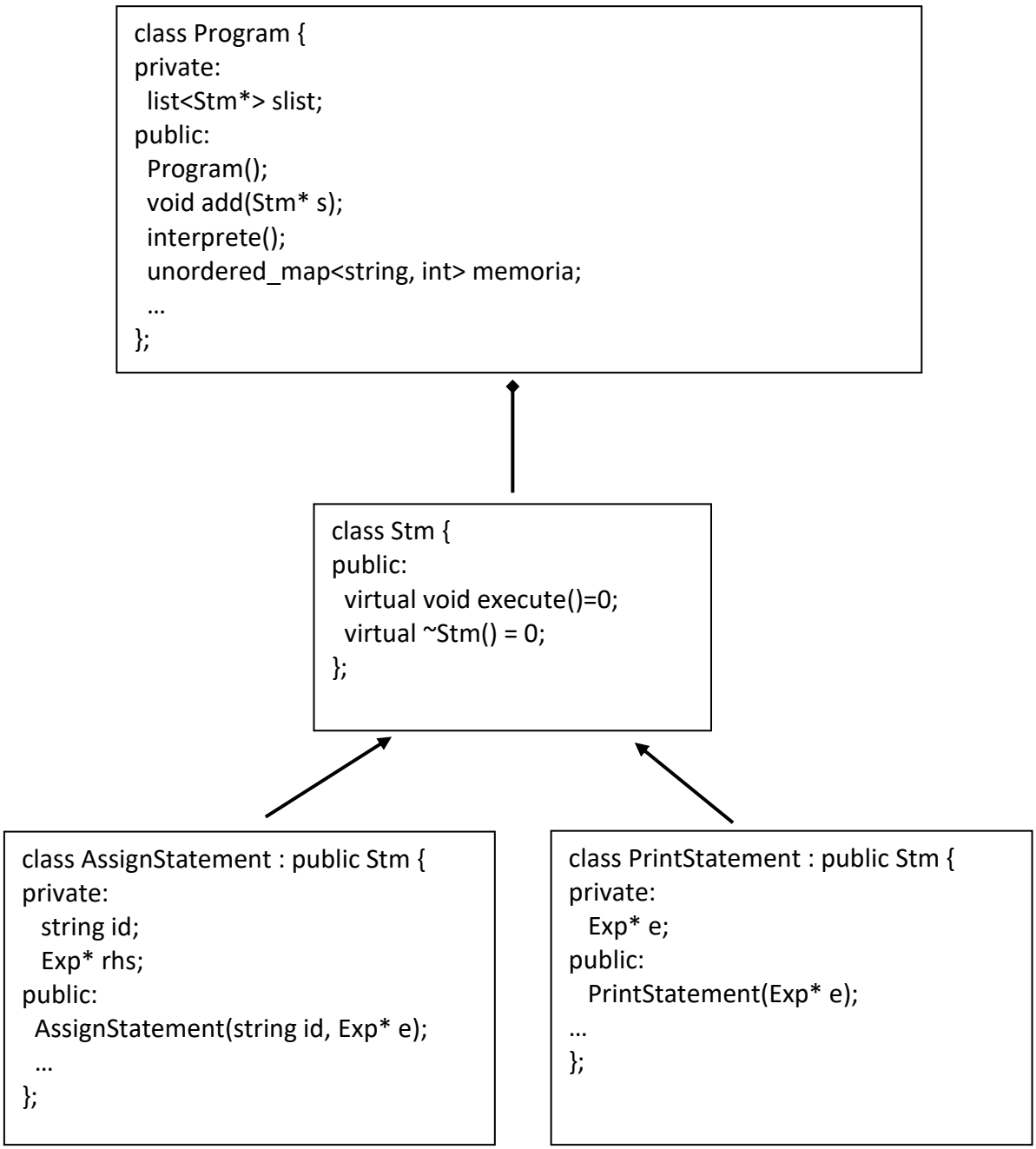
- **Agregar nuevos tokens**
  Se deben agregar los siguientes tokens a la lista existente:
  - ID: Para identificar variables.
  - PRINT: Para la palabra reservada print.
  - ASSIGN: Para el operador de asignación =.
  - PC: Para el punto y coma ;, que separa las sentencias.

- **Modificar el scanner para reconocer el token ID**
  El Scanner debe ser modificado para reconocer secuencias alfanuméricas que representan identificadores (nombres de variables). El scanner debe manejar correctamente estos identificadores y asignarles el token ID.

- **Verificar palabras reservadas**
  El Scanner debe comprobar si el lexema reconocido para ID corresponde a la palabra reservada print. En caso afirmativo, debe devolver el token PRINT en lugar de ID, para diferenciar entre un identificador y la palabra clave print.

- **Verificación con test_scanner(&scanner)**
  Utilice la función test_scanner(&scanner) para realizar pruebas y asegurarse de que el Scanner está reconociendo correctamente los nuevos tokens. Esta función debe imprimir todos los tokens reconocidos, lo que permitirá verificar que tanto los identificadores como las palabras reservadas y los operadores son correctamente identificados.

- **Crear nuevas clases para el árbol de análisis sintáctico (AST)**


class Program {
private:
list<Stm*> slist;
public:
Program();
void add(Stm* s);
interprete();
unordered_map<string, int> memoria;
…
};
class Stm {
public:
virtual void execute()=0;
virtual ~Stm() = 0;
};
class AssignStatement : public Stm {
private:
string id;
Exp* rhs;
public:
AssignStatement(string id, Exp* e);
…
};
class PrintStatement : public Stm {
private:
Exp* e;
public:
PrintStatement(Exp* e);
…
};

- **Crear los métodos en el Parser**

```
Program* Parser::parseProgram() {
        Program* p = new Program();
        p->add(parseStatement());
        while(match(Token::PC)) {
                p->add(parseStatement());
        }
         return p;
}
```

```
Stm* Parser::parseStatement() {
  Stm* s = NULL;
  Exp* e;
  if (match(Token::ID)) {
        string lex = previous->lexema;
        if (!match(Token::ASSIGN)) {
                exit(0);
        }
        s = new AssignStatement(lex, parseExpression());
  }
  else if (match(Token::PRINT)) {
        if (!match(Token::LPAREN)) {
                exit(0);
        }
        e = parseExpression();
        if (!match(Token::RPAREN)) {
                exit(0);
        }
        s = new PrintStatement(e);
  }
   else {
        cout << "No se encontro Statement" << endl;
        exit(0);
  }
  return s;
}
```

- **Modifique Exp* Parser::parseFactor**.
  El método parseFactor debe ser modificado para que pueda manejar variables (identificadores) además de números y expresiones entre paréntesis. La principal tarea es reconocer cuando se encuentra una variable (id), de manera que se

pueda buscar su valor en la memoria durante la evaluación.

Verifique que las expresiones:

a. x=6; y=7; print(x+y)
b. x=2; y=2*x; print(x+y)

- **Implemente el método execute de las sentencias (Stmt)**
  Este método se encarga de ejecutar la acción correspondiente según el tipo de sentencia (asignación o impresión).

  - Asignación (id = Exp)
    El valor de la expresión debe ser evaluado y almacenado en la memoria (un unordered_map<string, int> memoria), donde la clave será el nombre de la variable (id) y el valor será el resultado de la evaluación de la expresión.

  - Impresión (print(Exp))
    El método debe evaluar la expresión y mostrar su resultado en pantalla.