Práctica 3 -Semántica y generación de código. El compilador completo

PL3 - Memoria

Rubén Barroso, Marcos Fuster, Cristian Márquez

Primera parte: generando código intermedio en Jasmin
Segunda parte: compilando MiniB

Primera parte: generando código intermedio en Jasmin

Para esta primera parte, vamos a:

- Escribir en código intermedio Jasmin un programa para cada uno de los casos requeridos
- Generar una clase con Jasmin a partir de nuestro código
- Ver cómo sería su contraparte en código Java, traduciendo el .class de vuelta
- Comprobar su funcionamiento ejecutando la clase

Ejemplos

1.Programa principal vacío:

Código Jasmin:

```
.class public Vacio
.super java/lang/Object

.method public static main([Ljava/lang/String;)V

    ; reservar espacio en la pila, podria ser 0 aqui
    .limit stack 0

    ; terminar el main
    return

.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class Vacio {
    public static void main(String[] var0) {
    }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java Vacio
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class>
```

2.Programa principal que muestre por pantalla una cadena de texto:

Código Jasmin:

```
.class public Print
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
   .limit stack 100
   .limit locals 100

   getstatic java/lang/System/out Ljava/io/PrintStream;

   ldc "Hello World"

   invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
return

.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class Print {
   public static void main(String[] var0) {
      System.out.println("Hello World");
   }
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java Print
Hello World
```

3.Programa principal que multiplique dos números enteros y lo muestre por pantalla:

Código Jasmin:

```
.class public Product
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
   .limit stack 100
   .limit locals 100

   getstatic java/lang/System/out Ljava/io/PrintStream;
ldc 2
ldc 2
imul
   invokevirtual java/io/PrintStream/println(I)V
return

.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class Product {
   public static void main(String[] var0) {
      System.out.println(2 * 2);
   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java Product
4
```

4.Programa principal que muestre por pantalla el resultado de una operación lógica:

Código Jasmin:

```
.class public LogicOp
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100

    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 0
    ldc 1
    iand

    invokevirtual java/io/PrintStream/println(I)V
    return

.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class LogicOp {
   public static void main(String[] var0) {
      System.out.println(0 & 1);
   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java LogicOp
0
```

5.Programa principal que muestre por pantalla la concatenación de una cadena de texto y un número:

Código Jasmin:

```
.class public Concatenation
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100

    ; Create StringBuilder
    new java/lang/StringBuilder
    dup
    invokespecial java/lang/StringBuilder/<init>()V

    ; Append "Hello" to StringBuilder
    ldc "The answer is "
    invokevirtual java/lang/StringBuilder/append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    ; Append 42 to StringBuilder (convert to String)
    ldc 42
    invokestatic java/lang/String/valueOf(I)Ljava/lang/String;  ; Convert int to String
    invokevirtual java/lang/StringBuilder/append(Ljava/lang/String;)Ljava/lang/StringBuilder;

    ; Convert StringBuilder to String
    invokevirtual java/lang/StringBuilder/toString()Ljava/lang/String;

    ; Print the result
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

    return
.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class Concatenation {
   public static void main(String[] var0) {
      System.out.println("The answer is " + String.valueOf(42));
   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java Concatenation
The answer is 42
```

6.Programa principal que realice uno o varios if anidados, mostrando el resultado de cada uno:

Código Jasmin:

```
.class public IfNested
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 3
    .limit locals 3

    ldc 5
    istore_1

    ldc 10
    istore 2
    istore_1

    ldc 15            ; Load 15 onto the stack (c)
    istore 3          ; Store it in local variable 3 (c)

    ; First if statement: if (a < b)
    iload 1           ; Load a onto the stack
    iload 2           ; Load b onto the stack
    if_icmpge else1   ; If a >= b, jump to else1

    ; Nested if statement: if (b < c)
    iload 2           ; Load b onto the stack
    iload 3           ; Load c onto the stack
    if_icmpge else2   ; If b >= c, jump to else2

    ; If b < c, print "b is less than c"
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "b is less than c"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto end_if       ; Jump to end_if

else2:
    ; If b >= c, print "b is not less than c"
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "b is not less than c"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

end_if:
    goto end_outer_if  ; Jump to end_outer_if

else1:
    ; If a >= b, print "a is not less than b"
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "a is not less than b"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V

end_outer_if:
    return
.end method
```

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class IfNested {
   public static void main(String[] var0) {
      byte var1 = 5;
      byte var2 = 10;
      byte var3 = 15;
      if (var1 < var2) {
         if (var2 < var3) {
            System.out.println("b is less than c");
         } else {
            System.out.println("b is not less than c");
         }
      } else {
         System.out.println("a is not less than b");
      }

   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java IfNested
b is less than c
```

7.Bucle for:

Código Jasmin:

```
.class public For
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 10
    .limit locals 2

    ; Inicializamos el contador (i) a 0
    iconst_0            ; Cargar el valor 0 en la pila
    istore_1            ; Guardar el valor de 0 en la variable local 1 (i)

    ; Inicia el bucle for: para i = 0; i < 5; i++
    start_loop:     ; Etiqueta para el inicio del bucle
    iload_1             ; Cargar el valor de i
    bipush 5            ; Cargar el valor 5
    if_icmpge end_loop  ; Si i >= 5, salimos del bucle

    ; Imprimir el valor de i
    getstatic java/lang/System/out Ljava/io/PrintStream;  ; Obtener System.out
    iload_1             ; Cargar el valor de i
    invokevirtual java/io/PrintStream/println(I)V   ; Imprimir el valor de i

    ; Incrementar i (i++)
    iinc 1 1            ; Incrementar el valor de la variable local 1 (i) por 1

    goto start_loop     ; Volver al inicio del bucle

    end_loop:      ; Etiqueta para el final del bucle

    return              ; Terminar la ejecución del método
.end method

```

Código generado:

```
// Source code is decompiled from a .class file using FernFlower decompiler.
public class For {
   public static void main(String[] var0) {
      for(int var1 = 0; var1 < 5; ++var1) {
         System.out.println(var1);
      }

   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java For
0
1
2
3
4
```

8.Bucle while:

Código Jasmin:

```
.class public While
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 10
    .limit locals 2

    ; Inicializamos el contador (i) a 9
    bipush 9            ; Cargar el valor 9 en la pila
    istore_1            ; Guardar el valor de 9 en la variable local 1 (i)

    ; Inicia el bucle while: mientras i > 0
    start_loop:     ; Etiqueta para el inicio del bucle
    iload_1             ; Cargar el valor de i
    ifle end_loop       ; Si i <= 0, salimos del bucle

    ; Imprimir el valor de i
    getstatic java/lang/System/out Ljava/io/PrintStream;  ; Obtener System.out
    iload_1             ; Cargar el valor de i
    invokevirtual java/io/PrintStream/println(I)V   ; Imprimir el valor de i

    ; Decrementar i en 2 (i -= 2)
    iinc 1 -2           ; Decrementar el valor de la variable local 1 (i) por 2

    goto start_loop     ; Volver al inicio del bucle

    end_loop:      ; Etiqueta para el final del bucle

    return

.end method

```

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class While {
    public static void main(String[] var0) {
        for(int var1 = 9; var1 > 0; var1 -= 2) {
            System.out.println(var1);
        }

    }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java While
9
7
5
3
1
```

9. Llamada a una función sin parámetros ni devolución de resultado:

Código Jasmin:

```
.class public methodVacio
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
   .limit stack 100
   .limit locals 100


   ; Llamar al método funcVacio
   invokestatic methodVacio/funcVacio()V

   ; Terminar el main
   return


.end method


.method public static funcVacio()V

   .limit stack 100
   .limit locals 100

   getstatic java/lang/System/out Ljava/io/PrintStream;

   ldc "Esta función es inútil"

   invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
   return

.end method

```

Se ha añadido que haga un print de *Esta función es inútil* para poder observar el funcionamiento, aunque eso haga que la función no esté vacía.

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class methodVacio {
   public static void main(String[] var0) {
      funcVacio();
   }

   public static void funcVacio() {
      System.out.println("Esta funci\u00f3n es in\u00fatil");
   }
}
```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java methodVacio
Esta función es inútil
```

10.Llamada a una función devolviendo el resultado en un entero:

Código Jasmin:

```
.class public methodRet
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 10
    .limit locals 10

    ; Llamar al método funcRet
    invokestatic methodRet/funcRet()I

    ; Imprimir el valor de retorno
    getstatic java/lang/System/out Ljava/io/PrintStream;
    swap
    invokevirtual java/io/PrintStream/println(I)V

    ; Terminar el main
    return

.end method

.method public static funcRet()I
    .limit stack 10
    .limit locals 10

    ; Retornar el valor 10
    bipush 10
    ireturn

.end method
```

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class methodRet {
   public static void main(String[] var0) {
      System.out.println(funcRet());
   }

   public static int funcRet() {
      return 10;
   }
}

```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java methodRet
10
```

11.Llamada a una función devolviendo un parámetro:

Código Jasmin:

```
.class public methodParam
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 2
    .limit locals 1

    ; Cargar el número 42 en la pila
    bipush 42

    ; Llamar al método funcPatam con el número como argumento
    invokestatic methodParam/funcParam(I)V

    ; Terminar el main
    return
.end method


.method public static funcParam(I)V
    .limit stack 2
    .limit locals 1

    ; Obtener System.out para imprimir
    getstatic java/lang/System/out Ljava/io/PrintStream;

    ; Cargar el número recibido (primer argumento)
    iload_0

    ; Llamar a println para imprimir el número
    invokevirtual java/io/PrintStream/println(I)V

    ; Terminar el método
    return
.end method

```

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class methodParam {
   public static void main(String[] var0) {
      funcParam(42);
   }

   public static void funcParam(int var0) {
      System.out.println(var0);
   }
}

```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java methodParam
42
```

12.Llamada a una función pasando varios parámetros:

Código Jasmin:

```
.class public methodParams
.super java/lang/Object

.method public static main([Ljava/lang/String;)V
    .limit stack 10
    .limit locals 2

    ; Llamar a la función funcParams con un String y un int
    ldc "Número:"   ; Cargar el String
    bipush 42                   ; Cargar el entero 42
    invokestatic methodParams/funcParams(Ljava/lang/String;I)V

    return
.end method

.method public static funcParams(Ljava/lang/String;I)V
    .limit stack 10
    .limit locals 3

    ; Local 0: String (argumento)
    ; Local 1: int (argumento)

    ; Imprimir el String
    getstatic java/lang/System/out Ljava/io/PrintStream;  ; Obtener System.out
    aload 0                                              ; Cargar el String
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V  ; Imprimir

    ; Imprimir el entero
    getstatic java/lang/System/out Ljava/io/PrintStream;  ; Obtener System.out
    iload 1                                              ; Cargar el int
    invokevirtual java/io/PrintStream/println(I)V        ; Imprimir

    return
.end method

```

Código generado:

```java
// Source code is decompiled from a .class file using FernFlower decompiler.
public class methodParams {
   public static void main(String[] var0) {
      funcParams("N\u00famero:", 42);
   }

   public static void funcParams(String var0, int var1) {
      System.out.println(var0);
      System.out.println(var1);
   }
}

```

Resultado:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p1\class> java methodParams
Número:
42
```

Segunda Parte: compilando MiniB

Nivel Básico

La estructura de tabla de datos implementada y su gestión.

Para la tabla de datos se ha utilizado una clase *SymbolTable* que almacena un diccionario con los identificadores y su índice, un contador del número de símbolos en la tabla y una variable para comprobar el error.

```
self.symbols = {}
self.var_count = 0
self.error = False
```

En *SymbolTable* se han introducido los siguientes métodos:

- **add(name:str ,value) -> int, any :** Agrega un nuevo símbolo a la tabla, devuelve el índice y avisa si la variable ya ha sido declarada.
- **get(self, name: str) -> int, any:** Devuelve el valor e índice de un símbolo, si existe, si no lanza un error.
- **get_by_index(self, index: int): -> str, any:** Devuelve el nombre de un símbolo, si existe, si no lanza un error.
- **mod(self, name: str, value): -> int:** Modifica el valor de un símbolo, si existe, si no lanza un error. Devuelve el índice de la variable.

En el archivo donde tratamos los visitors, *Visitor.py,* se crea una instancia de *SymbolTable* a modo de tabla de símbolos.

El visitor es responsable de recorrer el árbol de sintaxis abstracta generado por el análisis del código fuente y de producir el código intermedio para jasmin.

Entonces cada vez que necesitamos añadir u obtener información de la tabla de símbolos llamamos a los métodos apropiados de la instancia *tabla* (Nuestra tabla de símbolos en Visitors) de la clase *SymbolTable.*

Por ejemplo se usan add(),get() y mod() en:

- En **visitLet()**, ya que LET se usa para declarar una variable y por lo tanto se quiere almacenar su identificador y valor en la tabla de símbolos, es decir usando:

```
def visitLet(self, ctx: MiniBParser.LetContext):
    """
    Declara una variable, le asigna el último valor en pila,
    cena en la siguiente posición de locals.

    Guardamos dicha posición en la tabla de simbolos.
    """
    var_name = ctx.ID().getText()
    var_value = self.visit(ctx.exp)

    self.try_ID(ctx.exp, var_value)

    var_index = self.tabla.add(var_name, var_value)

    self.store_var(var_index, var_value)
```

- En **visitIdExpression()**, para comprobar si existe dicho identificador y obtener su valor en caso de que exista:

```
def visitIdExpression(self, ctx: MiniBParser.IdExpressionContext):
    """
    Carga el valor de la variable en la cima del stack.
    """
    var_name = ctx.ID().getText()
    _, var_value = self.tabla.get(var_name)
    # print("en id", var_name, var_value)
    # self.load_var(var_index, var_value)
    return var_value
```

- En **visitOp()**, ya que es llamado en el caso de modificar una variable ya creada y por lo tanto hay que modificar el valor de dicha variable en la tabla de símbolos:

```python
def visitOp(self, ctx: MiniBParser.OpContext):
    """
    Asigna un valor a una variable, y la almacena
    en su posición de locals.
    """
    # print("En op: ", ctx.ID().getText())
    var_name = ctx.ID().getText()
    var_value = self.visit(ctx.exp)

    var_index = self.tabla.mod(var_name, var_value)
    self.try_ID(ctx.exp, var_value)
    self.store_var(var_index, var_value)
```

No almacenamos los tipos de las variables en la tabla de símbolos ya que al usar Python, los tipos los tratamos de forma **implícita** ya dentro del código de Visitor. Por ejemplo en el caso de visitPrint() hacemos un match para comprobar el tipo del valor a hacer print, o al guardar o cargar valores de la pila:

```python
def visitPrint(self, ctx: MiniBParser.PrintContext):
    """
    Imprime resultado de una expresión
    (lo que se encuentre en la cima del stack)
    """
    self.add_instruction("getstatic java/lang/System/out Ljava/io/PrintStream;")

    value = self.visit(ctx.exp)

    is_op = False
    try:
        is_op = bool(ctx.exp.op)
    except Exception:
        pass

    self.try_ID(ctx.exp, value, not is_op)
    printtype = ""

    match value:
        case int():
            printtype = "I"
        case float():
            printtype = "F"
        case str():
            printtype = "Ljava/lang/String;"
        case bool():
            printtype = "Z"
        case list():
            printtype = "Ljava/util/List;"
    self.add_instruction(f"invokevirtual java/io/PrintStream/println({printtype})V")
```

```python
def store_var(self, var_index, var_value):
    """
    Guarda un valor en la pila, con el tipo de dato adecuado.
    """
    match var_value:
        case str() | None:
            self.add_instruction(f"astore_{var_index}")
        case int() | bool():
            self.add_instruction(f"istore_{var_index}")
        case float():
            self.add_instruction(f"fstore_{var_index}")

def load_var(self, var_index: int, var_value):
    """
    Carga una variable en la pila con el tipo adecuado.
    """
    match var_value:
        case str() | None:
            self.add_instruction(f"aload_{var_index}")
        case int() | bool():
            self.add_instruction(f"iload_{var_index}")
        case float():
            self.add_instruction(f"fload_{var_index}")
```

Los errores tratados en la parte básica.

En la práctica, la gestión de errores se implementa principalmente en *SymbolTable* ,*Visitor* y *Main.*

SymbolTable

En esta clase se gestionan los errores de guardado, carga y modificación de variables en la tabla de símbolos:

Al acceder a una variable con el método **get**, se verifica si la variable existe en la tabla. Si la variable no ha sido declarada, se imprime un error indicando que la variable no existe. Esto evita referencias a variables inexistentes que den errores en el tiempo de ejecución.

Esto también ocurre al modificar variables. Al modificar una variable con el método mod, se verifica si la variable está presente en la tabla. Si la variable no existe, se genera un error, asegurando que solo se modifiquen variables previamente declaradas.

Si ocurre un error, se pone **error = True** en la tabla de símbolos.

Además aunque no es un error como tal, al agregar una nueva variable mediante el método add, se verifica si el nombre ya existe en la tabla de símbolos. Si la variable ya ha sido declarada, se imprime una advertencia indicando que la variable ha sido declarada más de una vez.

Visitor

En esta clase se hace detección de errores semánticos durante la visita de los nodos del árbol y se evalúa la corrección semántica de las operaciones.

Se tiene la variable **error** inicializada a False. Esta es puesta a True cuando se detecta un error en el visitor.

Se verifican tipos de datos, operadores adecuados y estructuras válidas, generando errores cuando se detectan inconsistencias.

Propagación de errores de la tabla de símbolos: El visitor utiliza la tabla de símbolos, instancia de *SymbolTable,* para acceder y modificar variables. Si la tabla de símbolos reporta un error (por ejemplo, al intentar acceder a una variable no declarada), visitor captura este error.

Main (Gestión de errores y la explicación de su funcionamiento en general)

Main es responsable de coordinar el proceso completo de compilación y ejecución del código fuente, y maneja los errores que pueden ocurrir en cada etapa haciendo un print de dichos errores en la etapa en la que se produjeran.

El main tiene como posibles argumentos:

- **El archivo de entrada (argv[1])**
- **-o:** indica que el siguiente argumento es el output path del código intermedio generado para ser usado por **jasmin.** Por defecto si no se incluye, **ejemplo.j**
- **-m:** indica que el siguiente argumento es el modo de ejecución, si no se introduce ninguno, será **c** por default. Pueden estar incluidos cualquiera de los siguientes:
    - **v:** Indica que se genere el visitor
    - **c:** Indica que se compile el código a código intermedio para **jasmin** y este lo compile a bytecode para ser ejecutable.
    - **e:** Indica que al final se ejecute el bytecode generado por **jasmin**.

La ejecución del main sigue la siguiente estructura (suponiendo que en el comando para ejecutar el main se incluye **`-m ecv`** , si no, se deberían omitir las fases de ejecución correspondientes.):

- El main recibe el archivo de entrada y lo lee para iniciar el proceso de compilación. Si el archivo no existe o no puede ser leído, se genera un **error** indicando el problema.
- **Cuando el main encuentra un error en alguna etapa o fase del código, muestra que ha ocurrido dicho error y para su ejecución.**
- Utiliza **ANTLR** para generar el lexer (**MiniBLexer**) y el parser (**MiniBParser**).
- Si hay errores sintácticos o léxicos en el código fuente, el parser produce excepciones y el main captura estas excepciones y muestra mensajes de **error** detallados, indicando la ubicación y naturaleza del error en el código.
- Main crea una instancia de Visitor para recorrer el **AST** y generar el código intermedio. Como ya se ha mencionado, si durante la visita se producen errores semánticos el Visitor marca su variable **error** a **True** y main verifica dicha variable **error** para comprobar si el Visitor ha tenido algún error.
- Si no hay errores previos, el main toma el código generado por el Visitor y lo escribe en un archivo **.j .** Si ocurre un error de escritura (por ejemplo, falta de permisos o espacio en disco), **Main** informa al usuario.
- Invoca el ensamblador de **jasmin** para convertir el código en bytecode Java.
- Si **Jasmin** detecta errores en el código (como instrucciones inválidas o sintaxis incorrecta), **Main** captura la salida de error y la muestra para facilitar la corrección.
- El main intenta ejecutar el programa compilado utilizando la máquina virtual Java y si se produce un error en tiempo de ejecución captura la excepción además de dar un detalle del error, incluyendo la causa y posible ubicación en el código.

//Generación del código intermedio Jasmin para todos los ejemplos válidos de la PL2 (Junto a la ejecución del mismo)

Generación del código intermedio Jasmin para todos los ejemplos válidos de la PL2

for_continue:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 1
    istore_0
    FOR_START_0:
    iload_0
    ldc 5
    if_icmpgt FOR_END_0
    iload_0
    ldc 3
    if_icmpge ELSE_1
    goto FOR_CONTINUE_0
    goto ENDIF_1
    ELSE_1:
    ENDIF_1:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    iload_0
    invokevirtual java/io/PrintStream/println(I)V
    FOR_CONTINUE_0:
    iinc 0 1
    goto FOR_START_0
    FOR_END_0:
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
3
4
5
Programa ejecutado con éxito.
```

for_exit:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 1
    istore_0
    FOR_START_0:
    iload_0
    ldc 5
    if_icmpgt FOR_END_0
    iload_0
    ldc 3
    if_icmple ELSE_1
    goto FOR_END_0
    goto ENDIF_1
    ELSE_1:
    ENDIF_1:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    iload_0
    invokevirtual java/io/PrintStream/println(I)V
    FOR_CONTINUE_0:
    iinc 0 1
    goto FOR_START_0
    FOR_END_0:
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
1
2
3
Programa ejecutado con éxito.
```

for simple:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 1
    istore_0
    FOR_START_0:
    iload_0
    ldc 5
    if_icmpgt FOR_END_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    iload_0
    invokevirtual java/io/PrintStream/println(I)V
    FOR_CONTINUE_0:
    iinc 0 1
    goto FOR_START_0
    FOR_END_0:
    return
.end method
```

- Ejecución:

```
PS C:\Users\ruben\Desktop\Uni\4\Lenguaje\ProcLenguaje_PL3\p2> python ./Main.py ..\codigosMiniB\for_simple.bas -m ecv
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
1
2
3
4
5
Programa ejecutado con éxito.
```

functions:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object

.method public static ADD(II)I
    .limit stack 10
    .limit locals 3
    ; Load parameters into local variables

; Function body
iload_0
iload_1
iadd
    ireturn
.end method

.method public static SUB(II)I
    .limit stack 10
    .limit locals 3
    ; Load parameters into local variables

iload_0
iload_1
isub
    ireturn
.end method

.method public static MUL(II)I
    .limit stack 10
    .limit locals 3

    ; Load parameters into local variables
```

```
iload_0
iload_1
imul
    ireturn
.end method


.method public static DIV(II)I
    .limit stack 10
    .limit locals 3

    ; Load parameters into local variables

iload_0
iload_1
idiv
    ireturn
.end method
```

```
.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    ldc 2
    invokestatic MiniB/ADD(II)I
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    ldc 2
    invokestatic MiniB/SUB(II)I
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    ldc 2
    invokestatic MiniB/MUL(II)I
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    ldc 2
    invokestatic MiniB/DIV(II)I
    invokevirtual java/io/PrintStream/println(I)V
    return
.end method
```

- Ejecución:

```
3
-1
2
0
```

if_simple_true:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 2
    if_icmpge ELSE_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "one"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_0
    ELSE_0:
    ENDIF_0:
    ldc 1
    ifeq ELSE_1
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "two"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_1
    ELSE_1:
    ENDIF_1:
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
one
two
Programa ejecutado con éxito.
```

if_simple_false:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 2
    if_icmple ELSE_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "one"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_0
    ELSE_0:
    ENDIF_0:
    ldc 1
    ifeq ELSE_1
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "two"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_1
    ELSE_1:
    ENDIF_1:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "three"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
two
three
Programa ejecutado con éxito.
```

if_else:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    ldc 2
    if_icmpge ELSE_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "true"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_0
    ELSE_0:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "false"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    ENDIF_0:
    ldc 2
    ldc 3
    if_icmple ELSE_1
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "true"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    goto ENDIF_1
    ELSE_1:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "false"
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    ENDIF_1:
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
true
false
Programa ejecutado con éxito.
```

input:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc "Name: "
    invokevirtual java/io/PrintStream/print(Ljava/lang/String;)V
    new java/util/Scanner
    dup
    getstatic java/lang/System/in Ljava/io/InputStream;
    invokespecial java/util/Scanner/<init>(Ljava/io/InputStream;)V
    invokevirtual java/util/Scanner/nextLine()Ljava/lang/String;
    astore_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    invokevirtual java/io/PrintStream/println(Ljava/lang/String;)V
    return
.end method
```

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
Name:
```

- Ejecución:

```
Name: hola mundo
```

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
Name: hola mundo
hola mundo
Programa ejecutado con éxito.
```

let: NO FUNCIONA AUN line 1:4 no viable alternative at input 'LETstring'

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc "foo"
    astore_0
    ldc 123
    istore_1
    aconst_null
    astore_2
    return
.end method
```

- Ejecución:
- ```
  Checking if visitor generation is needed...
  No need to generate the visitor. Files are up to date.
  Compilando fuente...
  Guardando archivo en: ./ejemplo.j
  Compilando a bytecode usando Jasmin...
  Generated: MiniB.class
  Bytecode generado con éxito.
  Ejecutando programa...
  Programa ejecutado con éxito.
  ```

logical: NO FUNCIONA AUN line 6:9 no viable alternative at input 'PRINTn3AND'

- Código intermedio:
- Ejecución:

op: NO FUNCIONA AUN line 1:4 no viable alternative at input 'LETstring'

- Código intermedio:
- Ejecución:

print:
  ```

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 255.921875
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 21.5
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 511.162109375
    invokevirtual java/io/PrintStream/println(F)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
255.92188
21.5
511.1621
Programa ejecutado con éxito.
```

repeat:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    istore_0
    REPEAT_START_0:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    iload_0
    invokevirtual java/io/PrintStream/println(I)V
    iload_0
    ldc 1
    iadd
    istore_0
    iload_0
    ldc 5
    if_icmpne REPEAT_START_0
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
1
2
3
4
Programa ejecutado con éxito.
```

while:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    istore_0
    WHILE_START_0:
    iload_0
    ldc 5
    if_icmpge WHILE_END_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    iload_0
    invokevirtual java/io/PrintStream/println(I)V
    iload_0
    ldc 1
    iadd
    istore_0
    goto WHILE_START_0
    WHILE_END_0:
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
En comparison
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
1
2
3
4
Programa ejecutado con éxito.
```

operations:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    ldc 2
    iadd
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 2
    ldc 1
    isub
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 2
    ldc 3
    imul
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 4
    ldc 2
    idiv
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 7
    ldc 2
    idiv
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 5
    ldc 3
    irem
    invokevirtual java/io/PrintStream/println(I)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
3
1
6
2
3
2
Programa ejecutado con éxito.
```

-

error_sintáctico:

- Código intermedio: **Ninguno.** Da error sintáctico que es detectado por el parsec.
- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
line 1:4 no viable alternative at input 'LET='
```

error_tipo:

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc "1"
    astore_0
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 1
    istore_1
    iload_1
    ldc 1
    isub
    invokevirtual java/io/PrintStream/println(I)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
0
Programa ejecutado con éxito.
```

Nótese que funciona porque se ha elaborado una mejora en la cual si tratas de sumar dos strings se concatenan, pero si tratas de sumar una string a un valor numérico como float o int, entonces se intenta convertir primero el string a un valor numérico y de poder hacerse se ejecuta como tal. Es decir

que los números se pueden poner como strings o como ints o floats. Sin embargo si no se logra transformar el string a valor numérico, no dará error, sino que transforma el valor numérico a string y los concatena. Tratar de hacer sin embargo por ejemplo “1”+”2” resultará en “12”, no en 3.

Nivel Intermedio

Definición de funciones y subrutinas:

Operaciones de cadenas (copiar una cadena, concatenar cadenas, extraer cadenas, operar con caracteres, etc.):

Concatenar cadenas

No es necesario cambios en la gramática para implementar esta capacidad

Adición de otros tipos básicos más allá del entero y la cadena: como mínimo: flotante, booleano, array (permitiendo redimensionarse):

Float

Modificamos la regla del lexer:

```
NUMBER: ('0x' [0-9a-fA-F]+ ('.' [0-9a-fA-F]+)?)
      | ('0b' [01]+ ('.' [01]+)?)
      | ('0o' [0-7]+ ('.' [0-7]+)?)
      | ([0-9]+ ('.' [0-9]+)?)
      ;
```

Boolean

Los tipos booleanos han sido implementados mediante el uso de variables enteras, tales como 0 o 1. Este comportamiento se puede observar en la siguiente condición:

```
IF 1 THEN
    PRINT "true"
END
IF 0 THEN
    PRINT "false"
END
```

Con el siguiente resultado:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: Ejemplos\IF
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
true
false
Programa ejecutado con éxito.
```

Array

En Visual Basic un array se crea con DIM seguido de un identificador y entre paréntesis el tamaño del array que se va a crear (Ej. cosas(5)), sin embargo para hacer más fácil la distinción respecto de las llamadas a funciones, se ha usado corchetes. Es decir, por ejemplo, cosas[5] es una array de tamaño 5.

Para ello a la gramática se han añadido las siguientes reglas:

- En el lexer: DIM para definir un array y REDIM para redimensionarlo/redefinir su tamaño.

  ```
  DIM:        'DIM' | 'dim' ;
  REDIM:      'REDIM' | 'redim' ;
  ```

- En el parser:

```
expression: left=expression op=arithmeticOp right=expression #ArithmeticExpression
          | '(' expr=expression ')'    #ParenExpression
          | fun=functionCall      #FunctionCallExpression
          | NUMBER            #NumberExpression
          | STRING_LITERAL    #StringExpression
          | ID                 #IdExpression
          | ID '[' expr=expression ']' #ArrayAccessExpression
          ;
```

```
dimStmt:    DIM ID '[' exp=expression ']' #Dim
            ;

redimStmt:  REDIM ID '[' exp=expression ']' #Redim
            ;
```

```
arrayOpStmt: ID '[' exp1=expression ']' '=' (exp2=expression | cond=condition) #ArrayOp
           ;
```

```
statement:  letStmt
          | opStmt
          | arrayOpStmt
          | printStmt
          | inputStmt
          | ifStmt
          | forStmt
          | whileStmt
          | repeatStmt
          | keyStmt
          | dimStmt
          | redimStmt
          | functionDefStmt
          ;
```

Mejora: Hexadecimal, octal y binario

También se ha añadido como mejora la posibilidad de trabajar con números en hexadecimal, octal y binario añadiendo **0x**, **0o** y **0b** respectivamente al comienzo del número.

Para ello en la gramática se ha modificado la regla del lexer:

```
NUMBER: ('0x' [0-9a-fA-F]+ ('.' [0-9a-fA-F]+)?)
      | ('0b' [01]+ ('.' [01]+)?)
      | ('0o' [0-7]+ ('.' [0-7]+)?)
      | ([0-9]+ ('.' [0-9]+)?)
      ;
```

Nivel Avanzado

Análisis semántico (e.g. comprobación de tipos, añadir a la tabla de símbolos, etc.) de las incorporaciones anteriores

Concatenación de cadenas

Para concatenar cadenas se ha probado el siguiente código:

```
LET x = "hola"
LET y = "mundo"
print x + y
```

Resultado ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
holamundo
Programa ejecutado con éxito.
```

Generación del código intermedio correspondiente en Jasmin y ejecución del mismo

Float y operaciones mezcladas de strings y valores numéricos

operations (MEJORADO):

Para demostrar el funcionamiento de las operaciones con float se usará el siguiente codigo operations ajustado a las mejoras:

```
LET a = 1
LET b = 1.1

PRINT a+"2"
PRINT 2-b
PRINT 2*3.3
PRINT 4/2.2
PRINT 7/2.2
PRINT 5.3 MOD 3
```

A parte de el uso de floats también se ha habilitado como se menciona anteriormente la posibilidad de realizar operaciones con valores numéricos en forma string. Por lo tanto se observa como a+”2” es 3.

- Código intermedio:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 1
    istore_0
    ldc 1.1
    fstore_1
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 2
    istore_2
    iload_0
    iload_2
    iadd
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 2.0
    fload_1
    fsub
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 2.0
    ldc 3.3
    fmul
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 4.0
    ldc 2.2
    fdiv
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 7.0
    ldc 2.2
    fdiv
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 5.3
    ldc 3.0
    frem
    invokevirtual java/io/PrintStream/println(F)V
    return
.end method
```

- Resultado ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
3
0.9
6.6
1.8181818
3.181818
2.3000002
Programa ejecutado con éxito.
```

Boolean

Array

A la hora de redimensionar los arrays, se ha hecho de tal forma que el array mantenga los elementos que tenía antes en caso de pasar a un tamaño mayor al que tenía antes. Los elementos vacíos se inicializan a 0.

Para comprobar los arrays se han creado principalmente dos nuevos archivos.

- array_simple
  - Código de MiniB:

```
DIM numbers[3]
DIM numbers2[4]
FOR i = 0 TO 2
    numbers[i] = i * 2
NEXT
PRINT numbers[0]
PRINT numbers[1]
PRINT numbers[2]
numbers2[0] = 5
numbers[0]=numbers2[0]
PRINT numbers[0]
```

- Código intermedio generado:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 3
    newarray int
    astore_0
    ldc 4
    newarray int
    astore_1
    ldc 0
    ldc 0
    istore_2
    FOR_START_0:
    iload_2
    ldc 2
    if_icmpgt FOR_END_0
    aload_0
    iload_2
    iload_2
    ldc 2
    imul
    iastore
    FOR_CONTINUE_0:
    iinc 2 1
    goto FOR_START_0
    FOR_END_0:
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 0
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
```

```
    ldc 1
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 2
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    aload_1
    ldc 0
    ldc 5
    iastore
    aload_0
    ldc 0
    aload_1
    ldc 0
    iaload
    iastore
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 0
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
0
2
4
5
Programa ejecutado con éxito.
```

- array_redim
  - Código de MiniB:

```
DIM numbers[3]
numbers[0] = 1
numbers[1] = 2
numbers[2] = 3
REDIM numbers[5]
numbers[3] = 4
PRINT numbers[0]
PRINT numbers[1]
PRINT numbers[2]
PRINT numbers[3]
PRINT numbers[4]
```

  - Código intermedio generado:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    ldc 3
    newarray int
    astore_0
    aload_0
    ldc 0
    ldc 1
    iastore
    aload_0
    ldc 1
    ldc 2
    iastore
    aload_0
    ldc 2
    ldc 3
    iastore
    ldc 5
    newarray int
    astore_1
    iconst_0
    istore_2
    FOR_START_0:
    iload_2
    ldc 2
    if_icmpgt FOR_END_0
    aload_1
    iload_2
    aload_0
    iload_2
    iaload
    iastore
    FOR_CONTINUE_0:
    iinc 2 1
    goto FOR_START_0
    FOR_END_0:
    ldc 5
    newarray int
    astore_0
    iconst_0
    istore_3
```

```
    FOR_START_1:
    iload_3
    ldc 2
    if_icmpgt FOR_END_1
    aload_0
    iload_3
    aload_1
    iload_3
    iaload
    iastore
    FOR_CONTINUE_1:
    iinc 3 1
    goto FOR_START_1
    FOR_END_1:
    aload_0
    ldc 3
    ldc 4
    iastore
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 0
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 1
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 2
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 3
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    aload_0
    ldc 4
    iaload
    invokevirtual java/io/PrintStream/println(I)V
    return
.end method
```

- Ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
1
2
3
4
0
Programa ejecutado con éxito.
```

Mejora: Hexadecimal,octal y binario

Para comprobar esta mejora se ha creado el archivo printMejorado:

```
PRINT 0xFF.EC
PRINT 0b010101.10
PRINT 0o777.123
```

- Código intermedio generado:

```
.class public MiniB
.super java/lang/Object


.method public static main([Ljava/lang/String;)V
    .limit stack 100
    .limit locals 100
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 255.921875
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 21.5
    invokevirtual java/io/PrintStream/println(F)V
    getstatic java/lang/System/out Ljava/io/PrintStream;
    ldc 511.162109375
    invokevirtual java/io/PrintStream/println(F)V
    return
.end method
```

- Resultado de ejecución:

```
Checking if visitor generation is needed...
No need to generate the visitor. Files are up to date.
Compilando fuente...
Guardando archivo en: ./ejemplo.j
Compilando a bytecode usando Jasmin...
Generated: MiniB.class
Bytecode generado con éxito.
Ejecutando programa...
255.92188
21.5
511.1621
```