Guía de Desarrollo - Compilador Python to x86_64

Arquitectura de Desarrollo

Estructura del Código Fuente

```
src/
├── main/java/
│   ├── codegen/
│   │   └── CodeGenerator.java      # Generación de código ASM
│   └── parser/
│       ├── ASTBuilder.java         # Construcción del AST
│       ├── IndentationLexer.java   # Manejo de indentación
│       ├── Main.java               # Punto de entrada
│       └── ast/
│           ├── ASTNode.java        # Interfaz base
│           ├── ASTVisitor.java     # Patrón Visitor
│           ├── ASTPrinter.java     # Depuración AST
│           └── *Node.java          # Nodos específicos
├── main/antlr4/parser/             # Clases generadas por ANTLR
└── test/                           # Archivos de prueba Python
```

Stack Tecnológico

- **Lenguaje**: Java 8+
- **Parser Generator**: ANTLR 4.13.2
- **Target**: x86_64 Assembly (Linux ABI)
- **Build System**: Manual compilation
- **Testing**: Archivos de prueba en `src/test/`

Configuración del Entorno de Desarrollo

Prerrequisitos

1. **JDK 8 o superior**

   ```
   java -version  # Verificar instalación
   ```

2. **ANTLR 4.13.2**
   - Descargar `antlr-4.13.2-complete.jar`
   - Colocar en directorio `lib/`

3. **Editor recomendado**: VS Code con extensión Java

Setup del Workspace

1. **Compilar ANTLR Grammar**

```
cd grammar/
java -jar ../lib/antlr-4.13.2-complete.jar PythonSubset.g4 -visitor -o
../src/main/antlr4/parser/
```

2. **Compilar Proyecto Completo**

```
# Desde directorio raíz
javac -cp "lib/*" -d build/ src/main/java/**/*.java
src/main/antlr4/parser/*.java
```

3. **Verificar Setup**

```
java -cp "build:lib/*" parser.Main src/test/ejemplo.py
```

Flujo de Desarrollo

Agregar Nuevas Construcciones Sintácticas

1. Modificar Gramática ANTLR

Editar `grammar/PythonSubset.g4`:

```
// Ejemplo: Agregar condicional if
if_stmt
    : IF expression COLON NEWLINE suite (ELSE COLON NEWLINE suite)?
    ;

compound_stmt
    : for_stmt
    | while_stmt
    | if_stmt  // Agregar nueva regla
    ;
```

2. Regenerar Parser

```
cd grammar/
java -jar ../lib/antlr-4.13.2-complete.jar PythonSubset.g4 -visitor -o
../src/main/antlr4/parser/
```

3. Crear Nodo AST

src/main/java/parser/ast/IfNode.java:

```
public class IfNode implements ASTNode {
    private ASTNode condition;
    private List<ASTNode> thenBody;
    private List<ASTNode> elseBody; // puede ser null

    public IfNode(ASTNode condition, List<ASTNode> thenBody, List<ASTNode>
elseBody) {
        this.condition = condition;
        this.thenBody = thenBody;
        this.elseBody = elseBody;
    }

    @Override
    public <T> T accept(ASTVisitor<T> visitor) {
        return visitor.visit(this);
    }

    // Getters...
}
```

4. Actualizar ASTVisitor Interface

src/main/java/parser/ast/ASTVisitor.java:

```
public interface ASTVisitor<T> {
    // ... métodos existentes ...
    T visit(IfNode node);  // Agregar nuevo método
}
```

5. Implementar en ASTBuilder

src/main/java/parser/ASTBuilder.java:

```
@Override
public ASTNode visitIf_stmt(PythonSubsetParser.If_stmtContext ctx) {
    ASTNode condition = visit(ctx.expression());

    List<ASTNode> thenBody = new ArrayList<>();
    for (PythonSubsetParser.StatementContext stmt : ctx.suite(0).statement()) {
        thenBody.add(visit(stmt));
    }

    List<ASTNode> elseBody = null;
```

```
        if (ctx.suite().size() > 1) {  // hay else
            elseBody = new ArrayList<>();
            for (PythonSubsetParser.StatementContext stmt : ctx.suite(1).statement())
    {
                elseBody.add(visit(stmt));
            }
        }

        return new IfNode(condition, thenBody, elseBody);
    }
```

6. Implementar Generación de Código

src/main/java/codegen/CodeGenerator.java:

```
    @Override
    public String visit(IfNode node) {
        StringBuilder sb = new StringBuilder();
        String elseLabel = "else_" + labelCounter++;
        String endLabel = "end_if_" + labelCounter++;

        // Evaluar condición
        sb.append(node.getCondition().accept(this));
        sb.append("    cmp rax, 0\n");
        sb.append("    je ").append(elseLabel).append("\n");

        // Then body
        for (ASTNode stmt : node.getThenBody()) {
            sb.append(stmt.accept(this));
        }
        sb.append("    jmp ").append(endLabel).append("\n");

        // Else body (si existe)
        sb.append(elseLabel).append(":\n");
        if (node.getElseBody() != null) {
            for (ASTNode stmt : node.getElseBody()) {
                sb.append(stmt.accept(this));
            }
        }

        sb.append(endLabel).append(":\n");
        return sb.toString();
    }
```

7. Actualizar ASTPrinter para Depuración

src/main/java/parser/ast/ASTPrinter.java:

```
    @Override
    public String visit(IfNode node) {
        StringBuilder sb = new StringBuilder();
        sb.append("IfNode:\n");
        sb.append("  condition:
    ").append(node.getCondition().accept(this)).append("\n");
        sb.append("  then: [\n");
        for (ASTNode stmt : node.getThenBody()) {
            sb.append("    ").append(stmt.accept(this)).append("\n");
        }
        sb.append("  ]\n");
        if (node.getElseBody() != null) {
            sb.append("  else: [\n");
            for (ASTNode stmt : node.getElseBody()) {
                sb.append("    ").append(stmt.accept(this)).append("\n");
            }
            sb.append("  ]\n");
        }
        return sb.toString();
    }
```

Testing

Crear Archivo de Prueba

`src/test/test_if.py`:

```
x = 10
if x > 5:
    print("Mayor que 5")
else:
    print("Menor o igual que 5")
print("Fin")
```

Ejecutar y Verificar

```
java -cp "build:lib/*" parser.Main src/test/test_if.py
cat build/ejemplo.asm  # Verificar código generado
```

Convenciones de Código

Estilo Java

- **Nomenclatura**:
  - Clases: `PascalCase`

  - Métodos/variables: `camelCase`
  - Constantes: `UPPER_CASE`

- **Estructura de clases**:

```
public class ExampleNode implements ASTNode {
    // Fields privados
    private Type field;

    // Constructor
    public ExampleNode(Type field) {
        this.field = field;
    }

    // Getters
    public Type getField() { return field; }

    // Accept method (requerido por ASTNode)
    @Override
    public <T> T accept(ASTVisitor<T> visitor) {
        return visitor.visit(this);
    }
}
```

Estilo Assembly

- **Indentación**: 4 espacios para instrucciones
- **Labels**: Sin indentación, terminados en `:`
- **Comentarios**: `; Descripción`

```
section .text
global _start

_start:
    ; Inicialización
    mov rax, 1
    mov rbx, 42

loop_start:
    ; Cuerpo del ciclo
    cmp rax, 10
    jge loop_end
    inc rax
    jmp loop_start

loop_end:
    ; Salida
    mov rax, 60
    syscall
```

Depuración y Testing

Habilitar Modo Debug

Modifica `Main.java`:

```
public class Main {
    private static final boolean DEBUG = true;

    public static void main(String[] args) {
        // ... código existente ...

        if (DEBUG) {
            System.out.println("=== TOKENS ===");
            // Imprimir tokens

            System.out.println("=== AST ===");
            ASTPrinter printer = new ASTPrinter();
            System.out.println(program.accept(printer));

            System.out.println("=== ASSEMBLY ===");
        }

        // ... generación de código ...
    }
}
```

Casos de Prueba

Estructura recomendada para archivos de test:

```
# test_feature.py
# Caso simple
simple_case = 42

# Caso complejo
for i in range(5):
    if i % 2 == 0:
        print("Par")
    else:
        print("Impar")
```

Verificación de Output

```
# Compilar
java -cp "build:lib/*" parser.Main src/test/test_feature.py

# Ensamblar y ejecutar
```

```
nasm -f elf64 build/ejemplo.asm -o build/ejemplo.o
gcc build/ejemplo.o -o build/programa
./build/programa
```

Optimizaciones y Mejoras

Performance

1. **Reutilización de registros**
    - Actual: Stack-based evaluation
    - Mejora: Register allocation
2. **Eliminación de código muerto**
    - Detectar variables no usadas
    - Optimizar expresiones constantes
3. **Optimización de ciclos**
    - Loop unrolling para rangos conocidos
    - Strength reduction

Características Pendientes

1. **Funciones definidas por usuario**
2. **Arrays y listas**
3. **Condicionales if/else/elif**
4. **Operadores de asignación compuesta** (`+=`, `-=`)
5. **Import system**

Arquitectura de Extensión

Plugin System (Future)

```
public interface CodegenPlugin {
    boolean canHandle(ASTNode node);
    String generate(ASTNode node, CodegenContext context);
}
```

Target Backends

El diseño actual permite agregar backends alternativos:

- **x86_32**: 32-bit assembly
- **ARM64**: ARM assembly
- **LLVM IR**: Para optimizaciones avanzadas
- **JVM Bytecode**: Java virtual machine

Análisis Semántico

Futuras mejoras pueden incluir:

```
public class SemanticAnalyzer implements ASTVisitor<Void> {
    private SymbolTable symbolTable;
    private List<SemanticError> errors;

    public void analyze(ProgNode program) {
        // Type checking
        // Variable declaration checking
        // Function call validation
    }
}
```

Contribución al Proyecto

Pull Request Workflow

1. **Fork y clone** del repositorio
2. **Crear branch** para feature: `git checkout -b feature/if-statements`
3. **Implementar** siguiendo las convenciones
4. **Testing exhaustivo** con casos edge
5. **Documentar** cambios en este archivo
6. **Submit PR** con descripción detallada

Coding Standards

- **Unit tests** para cada nueva característica
- **Documentación** inline en código complejo
- **Backwards compatibility** cuando sea posible
- **Performance testing** para cambios críticos

Bug Reports

Template para reportar bugs:

```
## Descripción
[Descripción concisa del problema]

## Reproducir
1. Archivo de prueba: [adjuntar archivo.py]
2. Comando ejecutado: [comando completo]
3. Output actual: [resultado obtenido]
4. Output esperado: [resultado esperado]

## Entorno
- OS: [Windows/Linux/macOS]
```

```
- Java version: [version]
- ANTLR version: [version]
```