Manual de Usuario - Compilador Python to x86_64

Guía de Instalación

Requisitos del Sistema

- **Java**: JDK 8 o superior
- **ANTLR 4.13.2**: Para regeneración de parsers (opcional)
- **Sistema Operativo**: Windows/Linux/macOS
- **Ensamblador**: NASM (para ensamblar el código generado)
- **Enlazador**: GCC o similar (para crear ejecutables)

Instalación Paso a Paso

1. **Clonar o descargar el proyecto**

```
git clone <repository-url>
cd python2asm_ProgSistBase1
```

2. **Compilar el proyecto**

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

3. **Verificar instalación**

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

Uso del Compilador

Sintaxis del Comando

```
java -cp "build:lib/*" parser.Main <archivo_python>
```

Ejemplos de Uso

Ejemplo 1: Programa Simple

```
# archivo: test_simple.py
x = 10
```

```
y = 20
print(x + y)
```

Compilación:

```
java -cp "build:lib/*" parser.Main test_simple.py
```

Resultado: Genera `ejemplo.asm` en el directorio `build/`

Ejemplo 2: Ciclo For

```
# archivo: test_for.py
for i in range(3):
    print(i)
print("Fin del ciclo for")
```

Código generado:

```
section .data
    ; ... strings literales ...

section .text
    ; ... código del ciclo ...
```

Ejemplo 3: Ciclo While

```
# archivo: test_while.py
contador = 0
while contador < 5:
    print(contador)
    contador = contador + 1
```

Ejecución del Código Generado

1. **Ensamblar**

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

2. **Enlazar**

```
gcc build/ejemplo.o -o build/programa -lc
```

3. **Ejecutar**

```
./build/programa
```

Sintaxis Soportada

Variables y Asignaciones

```
# Tipos soportados
x = 42          # Enteros
mensaje = "Hola" # Strings
activo = True   # Booleanos
inactivo = False

# Asignaciones
variable = expresion
```

Expresiones Aritméticas

```
# Operadores soportados
suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b

# Precedencia respetada
resultado = 2 + 3 * 4  # = 14
```

Expresiones Lógicas

```
# Operadores de comparación
mayor = a > b
menor = a < b
mayor_igual = a >= b
menor_igual = a <= b
igual = a == b
diferente = a != b

# Operadores lógicos (completamente implementados)
y_logico = True and False  # False
o_logico = True or False   # True
```

```
negacion = not True        # False

# Expresiones complejas
resultado = (x > 5) and (y < 10) or (z == 0)
complejo = not (a == b) and (c != d)
```

Estructuras de Control

Ciclo For

```
# Sintaxis básica
for variable in range(numero):
    # cuerpo del ciclo
    instrucciones...

# Ejemplo
for i in range(10):
    print(i)
    x = i * 2
```

Ciclo While

```
# Sintaxis básica
while condicion:
    # cuerpo del ciclo
    instrucciones...

# Ejemplo
contador = 0
while contador < 5:
    print(contador)
    contador = contador + 1
```

Condicionales If/Elif/Else

```
# Sintaxis básica
if condicion:
    # cuerpo del if
    instrucciones...
elif otra_condicion:
    # cuerpo del elif
    instrucciones...
else:
    # cuerpo del else
    instrucciones...
```

```
# Ejemplo simple
x = 10
if x > 15:
    print("Mayor que 15")
elif x > 5:
    print("Mayor que 5")
else:
    print("5 o menor")

# Condicionales anidados
if x > 0:
    if x < 10:
        print("Entre 0 y 10")
    else:
        print("Mayor o igual a 10")
else:
    print("Menor o igual a 0")
```

Llamadas a Funciones

```
# Función print soportada
print(variable)
print("string literal")
print(expresion_aritmetica)

# Función range para ciclos for
range(5)      # 0 a 4
range(10)     # 0 a 9
```

Limitaciones Actuales

Características No Soportadas

- **Funciones definidas por usuario**: Solo `print()` y `range()` están implementados
- **Listas y estructuras de datos**: Solo variables escalares (int, string, bool)
- **Import statements**: Sin soporte para módulos
- **Range con argumentos**: Solo soporta `range(stop)`, no `range(start, stop, step)`
- **Tipos float/decimal**: Solo enteros, strings y booleanos
- **Operaciones sobre strings**: No se pueden concatenar o manipular strings
- **Asignación compuesta**: No soporta `+=`, `-=`, `*=`, etc.
- **Slicing**: No se pueden hacer operaciones de slicing

Restricciones de Sintaxis

- **Indentación**: Debe ser consistente (4 espacios o tabs, no mezclar)
- **Nombres de variables**: Solo letras, números y underscore (no pueden empezar con número)
- **Strings**: Comillas dobles o simples (`"texto"` o `'texto'`)
- **Comentarios**: Soportados con `#` pero son ignorados por el compilador

- **Print**: Solo acepta un argumento a la vez
- **Range**: Solo acepta la forma `range(stop)` con un único argumento

Solución de Problemas

Errores Comunes

Error de Indentación

```
Error: Token recognition error at: '    '
```

Solución: Verificar que la indentación sea consistente

Error de Sintaxis

```
Error: mismatched input 'palabra' expecting {...}
```

Solución: Revisar que la sintaxis coincida con la gramática soportada

Error de Compilación Java

```
Exception in thread "main" java.lang.ClassNotFoundException
```

Solución: Verificar el classpath incluya `lib/*` y `build/`

Archivos de Prueba Incluidos

Archivo	Propósito	Descripción
`src/test/ejemplo.py`	Prueba básica	Ciclo for simple
`src/test/test_while.py`	Ciclos while	Múltiples ejemplos de while
`src/test/test_for.py`	Ciclos for	Variaciones de for
`src/test/test_logical.py`	Operadores lógicos	And, or, not

Depuración

Habilitar Modo Verbose

Modifica `Main.java` para agregar:

```
System.out.println("AST: " + program.toString());
System.out.println("Generated ASM saved to: " + outputPath);
```

Inspeccionar AST Generado

Usa `ASTPrinter` para visualizar el árbol:

```
ASTPrinter printer = new ASTPrinter();
program.accept(printer);
```

Ejemplos Completos

Programa de Suma Acumulativa

```
# suma_acumulativa.py
total = 0
for i in range(10):
    total = total + i
print(total)
```

Contador con While

```
# contador.py
numero = 1
while numero <= 5:
    print("Contador:")
    print(numero)
    numero = numero + 1
print("Terminado")
```

Operaciones Lógicas

```
# logica.py
a = True
b = False
resultado1 = a and b
resultado2 = a or b
print(resultado1)  # False
print(resultado2)  # True
```

Próximos Pasos

Después de usar exitosamente el compilador:

1. **Examinar código ASM**: Revisar `build/ejemplo.asm` generado

2. **Ejecutar programa**: Seguir pasos de ensamblado y enlazado
3. **Experimentar**: Probar diferentes construcciones sintácticas
4. **Contribuir**: Ver `development.md` para extender funcionalidad