

Ejemplos de Código - Python to x86_64 Compiler

Casos de Uso Básicos

1. Variables y Asignaciones

Ejemplo Simple

```
# Archivo: basic_variables.py
x = 42
mensaje = "Hola Mundo"
activo = True

print(x)
print(mensaje)
print(activo)
```

ASM Generado:

```
section .data
str12345: db "Hola Mundo", 0x0A, 0

section .text
global _start

_start:
; x = 42
mov rax, 42
mov [rbp-8], rax

; mensaje = "Hola Mundo"
lea rax, [str12345]
mov [rbp-16], rax

; activo = True
mov rax, 1
mov [rbp-24], rax

; print(x)
mov rax, [rbp-8]
call print_int

; print(mensaje)
mov rax, [rbp-16]
call print_string

; print(activo)

```

```
mov rax, [rbp-24]
call print_bool
```

2. Expresiones Aritméticas

Operaciones Básicas

```
# Archivo: arithmetic.py
a = 10
b = 5

suma = a + b
resta = a - b
multiplicacion = a * b
division = a / b

print(suma)
print(resta)
print(multiplicacion)
print(division)
```

Expresiones Complejas

```
# Archivo: complex_expressions.py
x = 2
y = 3
z = 4

# Precedencia de operadores
resultado1 = x + y * z      # = 2 + (3 * 4) = 14
resultado2 = (x + y) * z    # = (2 + 3) * 4 = 20
resultado3 = x * y + z * 2  # = (2 * 3) + (4 * 2) = 14

print(resultado1)
print(resultado2)
print(resultado3)
```

3. Expresiones Lógicas y Comparaciones

```
# Archivo: logical_operations.py
a = 10
b = 5
c = 10

# Comparaciones
mayor = a > b      # True
```

```
menor = a < b      # False
igual = a == c     # True
diferente = a != b # True

# Operadores lógicos
y_logico = mayor and igual      # True and True = True
o_logico = menor or diferente   # False or True = True
negacion = not menor             # not False = True

print(mayor)
print(menor)
print(igual)
print(diferente)
print(y_logico)
print(o_logico)
print(negacion)
```

Estructuras de Control

4. Ciclos For

For Simple

```
# Archivo: simple_for.py
print("Contando del 0 al 4:")
for i in range(5):
    print(i)
print("Fin del conteo")
```

Output Esperado:

```
Contando del 0 al 4:
0
1
2
3
4
Fin del conteo
```

For con Cálculos

```
# Archivo: for_calculations.py
suma_total = 0

print("Calculando suma de 0 a 9:")
for numero in range(10):
```

```
suma_total = suma_total + numero
print("Número:")
print(numero)
print("Suma parcial:")
print(suma_total)

print("Suma final:")
print(suma_total)
```

For con Expresiones

```
# Archivo: for_expressions.py
print("Tabla del 2:")
for i in range(1, 6): # No soportado aún - usar range(5)
    resultado = i * 2
    print(i)
    print(" * 2 = ")
    print(resultado)
```

5. Ciclos While

While Básico

```
# Archivo: simple_while.py
contador = 0

print("Contando con while:")
while contador < 5:
    print("Contador:")
    print(contador)
    contador = contador + 1

print("Fin del while")
```

While con Condiciones Complejas

```
# Archivo: complex_while.py
x = 1
limite = 100

print("Potencias de 2 menores a 100:")
while x < limite:
    print(x)
    x = x * 2
```

```
print("Última potencia:")
print(x)
```

While con Booleanos

```
# Archivo: while_boolean.py
continuar = True
contador = 0

while continuar:
    print("Iteración:")
    print(contador)
    contador = contador + 1

    # Simular condición de parada
    if contador == 3: # Requiere implementar if
        continuar = False
```

6. Combinando Estructuras

For Anidado (Conceptual)

```
# Archivo: nested_loops.py
# NOTA: Anidamiento no completamente probado

print("Tabla de multiplicar (conceptual):")
for i in range(3):
    print("Tabla del:")
    print(i)
    for j in range(3):
        resultado = i * j
        print(resultado)
    print("---")
```

While y For Combinados

```
# Archivo: mixed_loops.py
print("Ciclos combinados:")

# For primero
suma = 0
for i in range(5):
    suma = suma + i

print("Suma del for:")
print(suma)
```

```
# Luego while
contador = suma
print("Contando hacia abajo:")
while contador > 0:
    print(contador)
    contador = contador - 1

print("Terminado")
```

Casos de Uso Avanzados

7. Simulación de Algoritmos

Factorial (con While)

```
# Archivo: factorial.py
numero = 5
factorial = 1
contador = 1

print("Calculando factorial de:")
print(numero)

while contador <= numero:
    factorial = factorial * contador
    print("Paso:")
    print(contador)
    print("Factorial parcial:")
    print(factorial)
    contador = contador + 1

print("Factorial final:")
print(factorial)
```

Fibonacci

```
# Archivo: fibonacci.py
n = 10
a = 0
b = 1
contador = 0

print("Serie Fibonacci:")
print(a)
print(b)

while contador < n:
```

```
siguiente = a + b
print(siguiente)
a = b
b = siguiente
contador = contador + 1
```

Búsqueda Secuencial (Simulada)

```
# Archivo: search_simulation.py
# Simular búsqueda en "array" de 10 elementos
objetivo = 7
encontrado = False
posicion = 0

print("Buscando el número:")
print(objetivo)

while posicion < 10 and not encontrado:
    # Simular valor en posición (posición + 1)
    valor_actual = posicion + 1

    print("Revisando posición:")
    print(posicion)
    print("Valor:")
    print(valor_actual)

    if valor_actual == objetivo:
        encontrado = True
        print("¡Encontrado en posición:")
        print(posicion)
    else:
        posicion = posicion + 1

if not encontrado:
    print("No encontrado")
```

8. Validaciones y Testing

Testing de Operadores

```
# Archivo: operator_test.py
print("== Test de Operadores ==")

# Aritméticos
a = 15
b = 4

print("Suma:")
```

```
print(a + b)      # 19

print("Resta:")
print(a - b)      # 11

print("Multiplicación:")
print(a * b)      # 60

print("División:")
print(a / b)      # 3 (división entera)

# Comparaciones
print("Mayor que:")
print(a > b)      # True

print("Menor que:")
print(a < b)      # False

print("Igual:")
print(a == b)      # False

print("Diferente:")
print(a != b)      # True
```

Testing de Precedencia

```
# Archivo: precedence_test.py
print("==> Test de Precedencia ==>")

# Aritmética
resultado1 = 2 + 3 * 4
print("2 + 3 * 4 =")
print(resultado1)  # Debería ser 14

# Con paréntesis
resultado2 = (2 + 3) * 4
print("(2 + 3) * 4 =")
print(resultado2)  # Debería ser 20

# Lógica
a = True
b = False
c = True

resultado3 = a and b or c
print("True and False or True =")
print(resultado3)  # Debería ser True

# Comparación y lógica
x = 5
y = 10
```

```
resultado4 = x < y and y > 0
print("5 < 10 and 10 > 0 =")
print(resultado4) # Debería ser True
```

Casos Edge y Limitaciones

9. Casos Límite

Números Grandes

```
# Archivo: large_numbers.py
grande = 1000000
print("Número grande:")
print(grande)

# Operaciones con números grandes
resultado = grande * 2
print("Doble:")
print(resultado)
```

Strings con Espacios

```
# Archivo: string_spaces.py
mensaje1 = "Hola mundo con espacios"
mensaje2 = "123 números en string"
mensaje3 = "" # String vacío

print(mensaje1)
print(mensaje2)
print(mensaje3)
```

Booleanos en Expresiones

```
# Archivo: boolean_expressions.py
verdadero = True
falso = False

# Booleanos como números (0 y 1)
suma_bool = verdadero + falso
print("True + False =")
print(suma_bool) # Debería ser 1

# En comparaciones
es_verdadero = verdadero == True
print("True == True =")
print(es_verdadero) # Debería ser True
```

10. Características No Soportadas

Ejemplos que NO Funcionan

```
# ESTOS EJEMPLOS NO FUNCIONAN AÚN

# 1. Condicionales if/else
if x > 5:
    print("Mayor")
else:
    print("Menor")

# 2. Funciones definidas por usuario
def mi_funcion(parametro):
    return parametro * 2

# 3. Listas
mi_lista = [1, 2, 3, 4, 5]

# 4. Range con múltiples parámetros
for i in range(1, 10, 2):
    print(i)

# 5. Asignación compuesta
x += 5
y *= 2

# 6. Múltiples asignaciones
a, b = 1, 2

# 7. Comentarios
# Este es un comentario
x = 5 # Comentario inline
```

Comandos de Testing

Ejecutar Ejemplos

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

# Ensamblar y ejecutar
nasm -f elf64 build/ejemplo.asm -o build/ejemplo.o
gcc build/ejemplo.o -o build/programa
./build/programa

# Testing batch (todos los ejemplos)
for file in src/test/*.py; do
```

```
echo "Testing $file"
java -cp "build:lib/*" parser.Main "$file"
if [ $? -eq 0 ]; then
    echo "✓ Compilación exitosa"
else
    echo "✗ Error en compilación"
fi
done
```

Output Esperado

Cada ejemplo debería:

1. **Compilar sin errores:** No excepciones Java
2. **Generar ASM válido:** Archivo `build/ ejemplo.asm` creado
3. **Ensamblar correctamente:** NASM sin errores
4. **Ejecutar y producir output:** Resultado coherente con código Python

Verificación de Resultados

```
# Ejemplo para verificar arithmetic.py
echo "Expected: 15, 5, 50, 2"
./build/programa
# Verificar que el output coincida
```

La documentación de ejemplos proporciona una guía completa para usuarios y desarrolladores sobre qué es posible hacer con el compilador actual y cómo estructurar código Python compatible.