Proyecto: Compilador de Fangless Python a C++

Descripción del Proyecto

El objetivo de este proyecto es diseñar e implementar un compilador que traduzca una versión simplificada de Python a C++. Este compilador se centrará en una gramática reducida de Python, eliminando características avanzadas y enfocándose en un subconjunto esencial del lenguaje para hacer el proyecto manejable en el contexto de un curso universitario. El proyecto incluye el desarrollo de un Lexer (analizador léxico) y un Parser (analizador sintáctico) utilizando PLY (Python Lex-Yacc

Aspectos Técnicos

- Herramienta de Desarrollo: Python, utilizando PLY (Python Lex-Yacc) para la implementación del Lexer y Parser.
- Idioma del Código y Documentación: Inglés.
- Control de Código: Git para versionado y colaboración.
- Colaboración: Equipos de hasta tres estudiantes.
- Fecha de Entrega: Viernes 15 de noviembre de 2024.
- Nomenclatura de Branches: TASK_#_BriefDescription.
- Buenas Prácticas de Código: Enfoque en código limpio, eficiente y mantenible. El código debe ser modular y fácilmente extensible para futuras mejoras.

Requerimientos del Proyecto

1. Lexer (Analizador Léxico)

1.1. Entrada y Salida

Entrada: Archivo de código fuente en Python (subconjunto definido).

Salida: Lista de tokens que representan palabras clave, identificadores, operadores, literales y otros elementos del lenguaje Python.

1.2. Características Incluidas en la Sintaxis

El Lexer debe reconocer los siguientes tipos de tokens:

1. **Palabras Clave (Keywords):**
    - Control de flujo: if, else, elif, while, for, break, continue, pass
    - Definición: def, return, class
    - Tipos booleanos: True, False
    - Otros: and, or, not
2. **Identificadores:**

- Nombres de variables, funciones y clases.
- **Reglas:**
  - Comienzan con una letra (a-z, A-Z) o un guion bajo _.
  - Pueden contener letras, dígitos (0-9) y guiones bajos _.
  - No pueden coincidir con palabras clave.

3. **Literales:**
   - **Numéricos:**
     - Enteros: Secuencia de dígitos (ej., 123)
     - Flotantes: Dígitos con punto decimal (ej., 123.45)
   - **Cadenas de Texto:**
     - Delimitadas por comillas simples ' o dobles " (ej., 'hola', "mundo")
     - Soportar secuencias de escape básicas (\n, \t, \\, \", \')
   - **Booleanos:**
     - True, False
4. **Operadores:**
   - **Aritméticos:** +, -, *, /, //, %, **
   - **Relacionales:** ==, !=, <, >, <=, >=
   - **Lógicos:** and, or, not
   - **Asignación:** =, +=, -=, *=, /=, %=, //=, **=
   - **Otros:** :, ,, ., (,), [,], {, }
5. **Delimitadores:**
   - Paréntesis: (,)
   - Corchetes: [,]
   - Llaves: {, }
   - Dos puntos: :
   - Coma: ,
   - Punto: .
   - Arroba: @ (solo si los decoradores simplificados se incluyen).
6. **Comentarios:**
   - **Comentarios de una sola línea:** Inician con # y se extienden hasta el final de la línea.
   - **Comentarios de múltiples líneas:** No se incluyen en esta versión simplificada.
7. **Indentación:**
   - Espacios en blanco y tabulaciones para definir bloques de código.
   - El Lexer debe generar tokens especiales para gestionar la indentación (INDENT, DEDENT) y finales de línea (NEWLINE).

1.3. Características Excluidas en la Sintaxis

El Lexer no debe reconocer ni procesar los siguientes elementos, ya que han sido eliminados del subconjunto de Python a implementar:

- Nested functions (funciones anidadas)

- Comprehensions (listas, diccionarios, conjuntos)
- Lambdas
- Functions as arguments (funciones como argumentos)
- Pointers
- Decorators
- Generators y yield
- Asynchronous Programming (async/await)
- Metaclasses
- Multiple Inheritance (herencia múltiple)
- Context Managers (with statement)
- Dynamic Attribute Access (getattr, setattr, etc.)
- Annotations (type hints)
- Imports en general (excepto módulos básicos predefinidos si se incluyen)
- Comprehensión de Clases
- try-finally, try-except-else
- F-strings

1.4. Manejo de Errores Léxicos

- **Caracteres Desconocidos:** Cualquier carácter que no pertenezca a los definidos en los tokens anteriores debe generar un error léxico.
- **Secuencias de Escape Inválidas:** En literales de cadenas, secuencias de escape que no sean reconocidas (\n, \t, \\, \", \') deben generar un error.
- **Indentación Incorrecta:** Manejar y reportar errores en la indentación que no sigan las reglas del subconjunto definido.

1.5. Testing del Lexer

- **Pruebas de Integración:**
  - Procesamiento de fragmentos completos de código Python simplificado.
  - Verificación de la correcta generación de tokens INDENT, DEDENT, y NEWLINE.

---

2. Parser (Analizador Sintáctico)

2.1. Entrada y Salida

- **Entrada:** Lista de tokens generada por el Lexer.
- **Salida:** Determinación de si la secuencia de tokens corresponde a un código Python válido según el subconjunto definido.

2.2. Características Incluidas en la Sintaxis

El Parser debe soportar las siguientes construcciones del subconjunto de Python definido:

1. **Estructura Básica del Programa:**
    - Secuencia de declaraciones y definiciones de funciones y clases.
2. **Control de Flujo:**
    - **Condicionales:**
        - `if` seguido de una condición y bloque de código.
        - Opcionalmente `elif` y `else` con sus respectivos bloques.
    - **Bucles:**
        - `while` seguido de una condición y bloque de código.
        - `for` con la sintaxis `for variable in iterable` y bloque de código.
    - **Instrucciones de Control de Bucle:**
        - `break`
        - `continue`
        - `pass`
3. **Funciones:**
    - **Definición:**
        - Uso de `def` para declarar funciones con nombre, parámetros (posicionales y por defecto) y cuerpo.
    - **Llamada:**
        - Invocación de funciones con argumentos posicionales y por defecto.
    - **Características Excluidas:**
        - No se permiten funciones anidadas, lambdas, ni funciones como argumentos.
4. **Clases y Objetos:**
    - **Definición de Clases:**
        - Uso de `class` para declarar clases con nombre, atributos y métodos.
    - **Herencia:**
        - Herencia simple (una sola clase base).
    - **Creación de Instancias:**
        - Instanciación de clases.
    - **Acceso a Atributos y Métodos:**
        - Acceso a atributos y métodos de instancias mediante la notación de punto (.).
5. **Expresiones:**
    - **Aritméticas:** `+`, `-`, `*`, `/`, `//`, `%`, `**`
    - **Relacionales:** `==`, `!=`, `<`, `>`, `<=`, `>=`
    - **Lógicas:** `and`, `or`, `not`
    - **Agrupación:** Uso de paréntesis para agrupar expresiones.
6. **Asignaciones:**
    - Asignación de valores a variables.
    - Asignación con operadores compuestos (`+=`, `-=`, etc.).
7. **Bloques de Código:**
    - Definidos por indentación.
    - Estructuras anidadas simples permitidas (por ejemplo, `if` dentro de `for`).
8. **Estructuras de Datos:**

- **Listas:** Creación, acceso por índice, modificación (append, remove, index).
- **Tuplas:** Creación y uso de tuplas inmutables.
- **Diccionarios:** Creación, acceso por clave, modificación (get, keys, values).
- **Conjuntos:** Creación y operaciones básicas (add, remove, union, intersection).

9. **Entrada y Salida:**
    - **Entrada:** Uso de input() para recibir datos del usuario.
    - **Salida:** Uso de print() para mostrar información al usuario.
10. **Manejo de Errores:**
    - **Excepciones:**
        - Uso de try-except para capturar y manejar excepciones básicas.
11. **Comentarios:**
    - **Soporte:** Ignorar comentarios de una sola línea (#).
12. **Tipos de Datos Básicos:**
    - **Numéricos:** int, float
    - **Booleanos:** bool
    - **Cadenas de Texto:** str
13. **Operaciones con Cadenas:**
    - Concatenación (+)
    - Repetición (*)
    - Acceso por índice (str[index])
    - Métodos básicos: lower(), upper(), find(), replace()
14. **Funciones de Conversión de Tipos:**
    - int(), float(), str(), bool()
15. **Iterables:**
    - Iteración sobre listas, tuplas y diccionarios usando bucles for.

2.3. Características Excluidas en la Sintaxis

El Parser no debe soportar ni reconocer las siguientes construcciones, ya que han sido eliminadas del subconjunto de Python a implementar:

- Nested functions (funciones anidadas)
- Comprehensions (listas, diccionarios, conjuntos)
- Lambdas
- Functions as arguments (funciones como argumentos)
- Pointers
- Decorators
- Generators y yield
- Asynchronous Programming (async/await)
- Metaclasses
- Multiple Inheritance (herencia múltiple)
- Context Managers (with statement)
- Dynamic Attribute Access (getattr, setattr, etc.)

- Annotations (type hints)
- Imports en general (excepto módulos básicos predefinidos si se incluyen)
- Comprehensión de Clases
- try-finally, try-except-else
- F-strings

2.4. Manejo de Errores Sintácticos

- **Estructura Incorrecta:** Detectar y reportar errores como sentencias mal formadas, bloques de código mal indentados, paréntesis o corchetes no balanceados.
- **Mensajes de Error Claros:** Indicar la línea y columna donde se detectó el error, así como una descripción del problema.
- **Recuperación de Errores:** Intentar continuar el análisis después de un error para detectar múltiples errores en una sola pasada cuando sea posible.

Características Adicionales

3. Manejo de Errores

Errores Léxicos y Sintácticos: El compilador debe manejar errores de manera robusta, permitiendo al usuario identificar y corregir problemas en el código Python original.

Mensajes Claros y Precisos: Los errores deben indicar la línea y columna del problema, así como una descripción clara del mismo.

4. Modularidad y Extensibilidad

Diseño Modular: Separar claramente las funcionalidades del Lexer y el Parser para facilitar la extensión a otras características de Python o adaptación a otros lenguajes de destino en el futuro.

5. Documentación Completa

Especificaciones Detalladas: Documentar cada componente del Lexer y Parser, incluyendo las reglas gramaticales y los tipos de tokens.

Ejemplos de Entrada y Salida: Proporcionar ejemplos claros de código Python simplificado y su correspondiente lista de tokens.

Guía de Usuario: Incluir instrucciones para la compilación, ejecución, y extensión del compilador.

Rúbrica de Evaluación

- **Completitud del Lexer y Parser (40 puntos):**
    - Identificación correcta de todos los tokens necesarios.

    - Análisis sintáctico preciso conforme al subconjunto definido de Python.
- **Calidad del Código (30 puntos):**
    - Código bien estructurado, modular y documentado.
    - Implementación eficiente del análisis léxico y sintáctico.
- **Manejo de Errores (15 puntos):**
    - Capacidad para detectar y reportar errores léxicos y sintácticos con mensajes claros y precisos.
- **Documentación (15 puntos):**
    - Calidad y claridad de la documentación técnica y de usuario.

Resumen de Características Incluidas y Excluidas

Incluidas:

- **Estructuras de Control:** if, elif, else, while, for, break, continue, pass
- **Definición de Funciones:** def con parámetros posicionales y por defecto
- **Definición de Clases:** class con herencia simple, métodos __init__ y __str__
- **Expresiones:** Aritméticas, lógicas, relacionales, agrupación con paréntesis
- **Asignaciones:** Con operadores compuestos
- **Estructuras de Datos:** Listas, tuplas, diccionarios, conjuntos
- **Entrada y Salida:** print(), input()
- **Manejo de Errores:** try-except básico
- **Comentarios:** Solo de una línea (#)
- **Tipos de Datos Básicos:** int, float, bool, str
- **Operaciones con Cadenas:** Concatenación, repetición, acceso por índice, métodos básicos
- **Funciones de Conversión de Tipos:** int(), float(), str(), bool()
- **Iterables:** Iteración sobre listas, tuplas y diccionarios con for

Excluidas:

- Nested functions (funciones anidadas)
- Comprehensions (listas, diccionarios, conjuntos)
- Lambdas
- Functions as arguments (funciones como argumentos)
- Pointers
- Decorators
- Generators y yield
- Asynchronous Programming (async/await)
- Metaclasses
- Multiple Inheritance (herencia múltiple)
- Context Managers (with statement)

- Dynamic Attribute Access (getattr, setattr, etc.)
- Annotations (type hints)
- Imports en general
- Comprehensión de Clases
- try-finally, try-except-else
- F-strings