Manual de Programador

Portada y Metadatos

- **Título**: “Manual de Programador – Micro-Compilador v1.0”
- **Autor**: Equipo de Desarrollo
- **Fecha**: 20 de mayo de 2025
- **Versión**: 1.0
- **Estado**: Borrador / Estable

Tabla de Contenidos

3. Guía de Estilo

- **Voz y tono**: Activo y directo, instrucciones en segunda persona plural para comandos, tercera persona para descripciones.
- **Formato**: Listados numerados para pasos, bloques de código para ejemplos, notas y advertencias destacadas.
- **Convenciones**: `snake_case` para funciones, `PascalCase` para clases, variables JSON en `camelCase`.

4. Glosario de Términos

- **AST**: Abstract Syntax Tree, estructura jerárquica que representa la sintaxis del código.
- **Token**: Unidad léxica mínima: palabras clave, identificadores, literales, operadores.
- **IR**: Intermediate Representation, código intermedio generado tras el análisis.
- **LL(1)**: Gramática y algoritmo de parsing de izquierdas con 1 símbolo de lookahead.

5. Quick Start

1. Clona el repositorio:

```
git clone https://github.com/Srinsoo/compilador_0.git && cd compilador_0
```

2. Crea y activa el entorno virtual:

```
python3 -m venv venv && source venv/bin/activate
```

3. Instala dependencias y compila un programa de ejemplo:

```
pip install -r requirements.txt
python interfaz.py --input ejemplo.src --output salida.ir --verbose
```

6. Arquitectura y Diagramas

- **Diagrama de Clases**: Representa `Lexer`, `Parser`, nodos de AST y relaciones.
- **Diagrama de Flujo**: Secuencia de pasos: lectura → tokenización → parsing → validación → generación de IR.

7. Estructura del Código

```
compilador_0/
├── docs/              ← Documentación fuente (reST)
├── lexer.py           ← Análisis léxico
├── parser.py          ← Análisis sintáctico y AST
├── interfaz.py        ← CLI/GUI
├── testing.py         ← Pruebas unitarias
```

```
├── requirements.txt  ← Dependencias
└── .gitignore        ← Archivos ignorados por Git
```

8. Entorno de Desarrollo

1. Clona y accede al repo.
2. Crea y activa entorno:

```
python3 -m venv venv
source venv/bin/activate
```

3. Instala dependencias:

```
pip install -r requirements.txt
```

4. Genera documentación:

```
cd docs
sphinx-quickstart
sphinx-apidoc -o source ../
make html
make latexpdf
```

9. Detalles de Implementación

- **lexer.py**: Clases `Lexer`, funciones `tokenize()`, manejo de errores por línea y columna.
- **parser.py**: Implementa parser LL(1), tablas de parsing, clases de nodos `Node`, `BinaryOp`, `Literal`.
- **interfaz.py**: Opciones:
  - `--input`, `--output`, `--verbose`, `--output-format ast|ir`
  - Plan de GUI con `tkinter` o `PyQt`.
- **testing.py**: Pruebas con `pytest`, fixtures y parametrización.

10. Pruebas, Métricas y QA

- Ejecuta pruebas:

```
pytest --maxfail=1 --disable-warnings -q
```

- Cobertura mínima: 90% (usar `coverage.py`).
- Reportes de cobertura:

```
coverage run -m pytest
coverage report
```

11. Integración Continua / Badges

Configura GitHub Actions (`.github/workflows/ci.yml`) para:

- Ejecutar pruebas y cobertura.
- Generar y desplegar documentación en Read the Docs.
- Badges en `README.md` para:
    - Build status
    - Coverage
    - Docs

12. Roadmap y Mantenimiento

- **v1.x**: GUI con PyQt, mejoras de optimización.
- **v2.0**: Soporte a nuevas construcciones de gramática, generación de código nativo.
- **Contribuciones**: Sigue semver MAJOR.MINOR.PATCH, abre issues con etiquetas (`enhancement`, `bug`, `docs`).

13. Licencia y Atribución

- **Licencia**: MIT License (archivo `LICENSE`).
- **Autores**: Maria Camila Arciniegas Pua, Orinson de Jesús Coba Núñez.