🧪 Fase 2 – Generación de Código Intermedio (Compiscript)

Este documento divide las tareas de la **Fase 2: Generación de Código Intermedio** en micro - tareas asignadas a varios integrantes del grupo.
Se busca claridad, responsabilidad individual y trazabilidad de commits en GitHub.

👤 Persona 1: **Expresiones y Operaciones Básicas (TAC Frontend)**

🔹 Micro - tareas

1. Diseñar la **estructura del TAC** (decidir: cuádruplas, tríplices, notación lineal).
2. Crear archivo base `TACGeneratorVisitor.py`.
3. Implementar generación TAC para:
    - Literales: enteros, booleanos, strings, `null`.
    - Operaciones aritméticas básicas: `+`, `-`.
    - Operaciones aritméticas avanzadas: `*`, `/`, `%`.
    - Expresiones agrupadas `(a + b) * c`.
4. Manejo de asignaciones:
    - Asignación simple `x = y`.
    - Asignación con operación `x = y + z`.
5. Agregar logs con los temporales generados (`t1, t2, …`).
6. Validar con ejemplos de expresiones simples.

👤 Persona 2: Operaciones Lógicas y Comparaciones

🔹 Micro - tareas

1. Implementar generación TAC para operaciones lógicas:
    - `&&`, `||`, `!` (con cortocircuito).
2. Implementar comparaciones:
    - `<`, `<=`, `>`, `>=`, `==`, `!=`.
3. Generar etiquetas (`L1`, `L2`) para control de flujo en expresiones booleanas.
4. Integrar resultados con los temporales de Persona 1.
5. Validar con casos de `if (a < b && c > d)`.

👤 Persona 3: Control de Flujo

🔹 Micro - tareas

1. Implementar TAC para estructuras condicionales:
    - `if` sin `else`.

- `if-else`.

2. Implementar TAC para bucles:
    - `while`.
    - `do-while`.
    - `for`.
    - `foreach`.
3. Manejo de sentencias:
    - `break`.
    - `continue`.
4. Validar con ejemplos de control de flujo anidado.

👤 Persona 4: Funciones y Procedimientos

🔹 Micro - tareas

1. Generar TAC para definición de funciones (prólogo y epílogo).
2. Implementar:
    - Llamada a función con parámetros (`param`, `call`).
    - Retorno de valor (`return`).
3. Integrar registros de activación:
    - Parámetros.
    - Variables locales.
    - Dirección de retorno.
4. Validar con ejemplos de funciones simples y recursivas (`factorial`).

👤 Persona 5: Tabla de Símbolos y Gestores

🔹 Micro - tareas

1. Revisar tabla de símbolos de la fase anterior.

2. Extender estructura para soportar:
    - Tipos de variable.
    - Offsets o direcciones relativas.
    - Etiquetas de funciones y variables globales.
    - Ámbitos anidados con enlace a padre.
3. Crear clase `TempManager`:
    - `new_temp()` para generar temporales únicos.
    - `free_temp(t)` para reciclar.
4. Crear clase `LabelManager`:
    - `new_label()` para generar etiquetas únicas (`L1`, `L2`).
5. Exportar tabla extendida a `symbol_table.json`.

👤 Persona 6: IDE y Ejecución

🔹 Micro‑tareas

1. Crear script `run_codegen.py`:
    - Ejecuta análisis completo (léxico, sintáctico, semántico).
    - Genera archivo TAC de salida.
2. Crear script `run_tests.sh`:
    - Corre todos los `.cps` en `tests/`.
    - Compara con `.tac` esperado.
3. Implementar **mini‑IDE**:
    - Opción CLI para editar y compilar.
    - (Opcional) interfaz web sencilla.
4. Mostrar en IDE: código fuente y TAC generado.

👤 Persona 7: Pruebas y Validación

🔹 Micro‑tareas

1. Diseñar casos de prueba exitosos:
    - Expresiones aritméticas.

- Control de flujo.
- Funciones simples.
- Objetos y clases básicos.

2. Diseñar casos de error semántico para validar robustez.
3. Automatizar validación de resultados (`expected.tac`).

👤 Persona 8: **Documentación y Entregables**

🔹 Micro - tareas

1. Escribir `docs/TAC_Spec.md`: definición del lenguaje intermedio con ejemplos.
2. Escribir `docs/SymbolTable.md`: estructura de la tabla de símbolos y RA.
3. Escribir `README_TAC.md`: cómo correr el compilador y generar TAC.
4. Documentar supuestos y limitaciones del diseño.
5. Confirmar que cada integrante tenga commits propios y claros.

📦 Estructura de Archivos Esperada

```
compiscript/
├── program/
│   ├── Compiscript.g4
│   ├── Driver.py
│   ├── tac_generator/
│   │   └── TACVisitor.py
│   └── tests/
│       ├── test_valid_01.cps
│       ├── test_valid_01.tac
│       └── ...
├── ide/
│   └── editor.py
├── docs/
│   ├── TAC_Spec.md
│   ├── SymbolTable.md
│   └── README_TAC_GENERATION.md
└── Dockerfile
```

✅ Con esta división, cada persona tiene pasos pequeños y claros, que permiten medir avances y asignar responsabilidades sin sobrecargar a nadie.