Spytective

Tagline: Investigate, Interrogate, Infiltrate.

Theme: Spytective Language is a fusion between **Detective deduction** and **Spy missions**.
Every program is a top-secret investigation; each line of code represents a lead, interrogation, or covert operation.
The syntax follows a C-like structure but replaces standard keywords with agent-style and detective-style vocabulary

Keyword Mapping

Concept	Keyword	Analogy	Example
Program start	crime_scene()	The main operation (entry point)	motive crime_scene() {...}
Function definition	motive	Mission or investigation function	motive decryptCode() {...}
Variable declaration	fine	Data declaration	fine clue= 7 ->>
Input	trigger	Get info from user	trigger agentName ->>
Output	report	Display info/result for user	report "Mission complete." ->>
If	investigate	Analyze or verify condition	investigate (clue > 7) {...}
Else	fallback	Alternative scenario	fallback { ... }
While	resist	Loop until condition	resist (clue != 0) {...}
Return	expose	Return data or conclusion	expose clue ->>
Comment	->>	Confidential notes	->> Top Secret

Data Types

Type	Meaning	Example
evidence	String	evidence criminal = ‘Bond’ ->>
fine	Integer	fine bondAge = 27 ->>
briberry	Float	briberry bondbalance = 600k ->>
intent	Boolean	intent missionReady = truth ->>
truth	True / 1	intent missionReady = truth ->>
alibi	False / 0	intent missionReady = alibi ->>

Syntax Examples

1.Hello World

```
motive crime_scene() {
    report "Welcome, Agent. Investigation initialized." ->>
    expose 0 ->>
}
```

2.Variables & Input

```
motive crime_scene() {
    evidence agentName ->>
    fine agentCode ->>

    report "Enter your agent name: " ->>
    trigger agentName ->>

    report "Enter your secret code: " ->>
    trigger agentCode ->>

    report "Agent " + agentName + " verified under code " + agentCode + "."  ->>
}
```

3.Investigate (Condition)

```
motive crime_scene() {
    fine clue = 9 ->>

    investigate (clue > 7) {
        report "Prime suspect detected!" ->>
    }
    fallback {
        report "No leads found. Keep investigating." ->>
    }
}
```

4.Function Example (Motive)

```
motive decryptCode(fine key) {
    report "Decrypting secret code..." ->>
    expose key * 2 ->>
}

motive crime_scene() {
    fine code = 21 ->>
    fine result = decryptCode(code) ->>

    investigate (result > 30) {
        report "Mission successful. Code cracked!" ->>
    }
    fallback {
        report "Mission failed. Encryption remains." ->>
    }
}
```

5.While loop (resistance Model)

```
motive crime_scene() {
    fine evidenceCount = 0 ->>

    resist (evidenceCount < 3) {
        report "Collecting evidence piece #" + evidenceCount ->>
        evidenceCount = evidenceCount + 1 ->>
    }

    report "All evidence gathered." ->>
}
```

6.Full Program Example

```
->> Case File #007: Deep Cover Investigation

motive analyzeEvidence(fine pieces) {
    report "Analyzing " + pieces + " pieces of evidence..." ->>

    resist (pieces > 0) {
        report "Processing evidence #" + pieces ->>
        pieces = pieces - 1 ->>
    }

    expose 1 ->>
}

motive crime_scene() {
    evidence agent = "Evelyn" ->>
    fine totalClues = 3 ->>

    report "Agent " + agent + " is on the case." ->>

    fine result = analyzeEvidence(totalClues) ->>

    investigate (result == truth) {
        report "Case closed successfully!" ->>
    }
    fallback {
        report "Case unresolved. Continue investigation." ->>
    }
}
```

Language Concept Summary

- Spytective merges the thrill of spy operations with the logic of detective investigations.
- It’s a story-driven programming language that turns logical structure into an undercover mission.

Core Design Goals

1. Make logic statements sound like mission steps and investigations.
2. Keep syntax intuitive for C/C++/Java learners.
3. Add narrative flavor to problem-solving and algorithms.