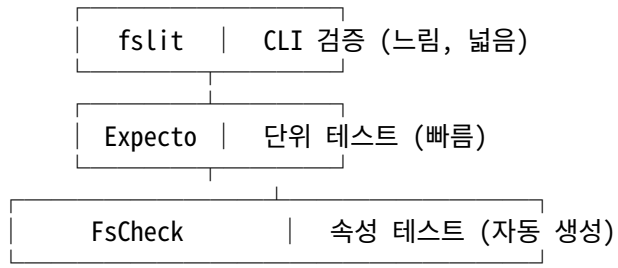
Appendix A: Testing

FunLang 인터프리터의 품질을 보장하기 위한 세 가지 테스트 전략을 소개한다. 각 접근법은 다른 목적에 최적화되어 있으며, 함께 사용할 때 가장 효과적이다.

개요

도구	목적	언제 사용
fslit	CLI 통합 테스트	회귀 방지, E2E 검증
Expecto	단위 테스트	모듈별 로직 검증
FsCheck	속성 기반 테스트	수학적 불변식 검증

테스트 피라미드:



---

fslit: 파일 기반 테스트

설치

```
dotnet tool install -g fslit
```

테스트 구조

```
tests/
├── Makefile
├── cli/              # 기본 평가 테스트
│   ├── 01-simple-add.flt
│   └── 02-precedence.flt
├── emit-tokens/      # --emit-tokens 테스트
├── emit-ast/         # --emit-ast 테스트
└── file-input/       # 파일 입력 테스트
```

.flt 파일 작성

기본 형식:

```
// 테스트 설명
// --- Command: dotnet run --project FunLang -- --expr "2 + 3"
// --- Output:
5
```

파일 입력 사용 (%input):

```
// 파일에서 프로그램 실행
// --- Command: dotnet run --project FunLang -- %input
// --- Input:
(2 + 3) * 4
// --- Output:
20
```

실행

```
# 전체 테스트
make -C tests

# 카테고리별
make -C tests cli
make -C tests emit-ast

# 단일 파일
fslit tests/cli/01-simple-add.flt
```

Makefile 예시

```
# tests/Makefile
.PHONY: all test cli emit-tokens emit-ast

all: test

test:
    @cd .. && fslit tests/

cli:
    @cd .. && fslit tests/cli/

emit-tokens:
    @cd .. && fslit tests/emit-tokens/

emit-ast:
    @cd .. && fslit tests/emit-ast/

check: build test

build:
    @cd .. && dotnet build
```

핵심 규칙

1. **한 파일 = 한 테스트** (fslit 제약)
2. **출력 정확 일치** (공백, 줄바꿈 포함)
3. **번호 접두사** (01-, 02-)로 순서 명시

Expecto: 단위 테스트

프로젝트 설정

```
# 테스트 프로젝트 생성
dotnet new console -lang F# -n FunLang.Tests -f net10.0
cd FunLang.Tests

# 패키지 추가
dotnet add package Expecto
dotnet add reference ../FunLang/FunLang.fsproj
```

테스트 코드

```
module FunLang.Tests

open Expecto
open FSharp.Text.Lexing

[<Tests>]
let lexerTests =
    testList "Lexer" [
        test "tokenizes number" {
            let lexbuf = LexBuffer<char>.FromString "42"
            let token = Lexer.tokenize lexbuf
            Expect.equal token (Parser.NUMBER 42) "should be NUMBER(42)"
        }
    ]

[<Tests>]
let parserTests =
    testList "Parser" [
        test "parses addition" {
            let lexbuf = LexBuffer<char>.FromString "2 + 3"
            let ast = Parser.start Lexer.tokenize lexbuf
            Expect.equal ast (Ast.Add(Ast.Number 2, Ast.Number 3)) ""
        }
    ]

[<Tests>]
let evalTests =
    testList "Eval" [
        test "evaluates addition" {
            let expr = Ast.Add(Ast.Number 2, Ast.Number 3)
            Expect.equal (Eval.eval expr) 5 ""
        }
    ]

[<EntryPoint>]
let main argv =
    runTestsWithCLIArgs [] argv <| testList "All" [
        lexerTests
```

```
        parserTests
        evalTests
    ]
```

실행

```
# 전체 테스트
dotnet run --project FunLang.Tests

# 필터링
dotnet run --project FunLang.Tests -- --filter "Lexer"
```

주요 Expect 함수

함수	용도
`Expect.equal actual expected msg`	동등성
`Expect.isTrue condition msg`	불린
`Expect.throws<ExnType> (fun () -> ...) msg`	예외
`Expect.isSome option msg`	Option 값

FsCheck: 속성 기반 테스트

설정

```
dotnet add package FsCheck
dotnet add package Expecto.FsCheck
```

속성 테스트 작성

"모든 입력에 대해 이 성질이 유지된다"를 선언한다.

```
open Expecto
open FsCheck

[<Tests>]
let propertyTests =
    testList "Properties" [
        // 숫자는 그대로 평가
        testProperty "number evaluates to itself" <| fun (n: int) ->
            Eval.eval (Ast.Number n) = n

        // 덧셈 교환법칙
        testProperty "addition is commutative" <| fun (a: int) (b: int) ->
            let left = Eval.eval (Ast.Add(Ast.Number a, Ast.Number b))
            let right = Eval.eval (Ast.Add(Ast.Number b, Ast.Number a))
            left = right
```

```
        // 0은 덧셈 항등원
        testProperty "zero is additive identity" <| fun (n: int) ->
            Eval.eval (Ast.Add(Ast.Number n, Ast.Number 0)) = n

        // 1은 곱셈 항등원
        testProperty "one is multiplicative identity" <| fun (n: int) ->
            Eval.eval (Ast.Multiply(Ast.Number n, Ast.Number 1)) = n

        // 이중 부정 = 원래 값
        testProperty "double negation is identity" <| fun (n: int) ->
            Eval.eval (Ast.Negate(Ast.Negate(Ast.Number n))) = n
    ]
```

조건부 속성

특정 조건에서만 성립하는 속성:

```
testProperty "division by non-zero" <| fun (a: int) (b: int) ->
    b <> 0 ==> lazy (
        // 0이 아닐 때만 테스트
        let result = Eval.eval (Ast.Divide(Ast.Number a, Ast.Number b))
        result = a / b
    )
```

실패 시 출력

FsCheck는 실패 시 **최소 반례**를 찾아준다:

```
Falsifiable, after 42 tests (3 shrinks):
Original: (1073741824, 1073741824)
Shrunk: (1, 2147483647)
```

Examples

CLI 테스트 추가

새 기능 구현 후 회귀 테스트 추가:

```
# 1. 실제 출력 확인
$ dotnet run --project FunLang -- --expr "let x = 5 in x + 1"
6

# 2. 테스트 파일 생성
$ cat > tests/variables/01-let-simple.flt << 'EOF'
// Test: Simple let binding
// --- Command: dotnet run --project FunLang -- --expr "let x = 5 in x + 1"
// --- Output:
6
EOF
```

```
# 3. 테스트 실행
$ make -C tests variables
PASS: tests/variables/01-let-simple.flt
Results: 1/1 passed
```

전체 테스트 실행

```
# fslit 통합 테스트
$ make -C tests
PASS: tests/cli/01-simple-add.flt
PASS: tests/cli/02-precedence.flt
...
Results: 21/21 passed

# Expecto 단위 테스트
$ dotnet run --project FunLang.Tests
[EXPECTO] 12 tests run - 12 passed, 0 failed

# 모두 통과 확인
```

테스트 실패 디버깅

```
# 토큰 확인
$ dotnet run --project FunLang -- --emit-tokens --expr "2 + 3"
NUMBER(2) PLUS NUMBER(3) EOF

# AST 확인
$ dotnet run --project FunLang -- --emit-ast --expr "2 + 3"
Add (Number 2, Number 3)
```

테스트 워크플로우

개발 중 (TDD)

```
# 1. 실패하는 테스트 작성
# 2. 구현
# 3. 테스트 통과 확인
dotnet run --project FunLang.Tests -- --filter "NewFeature"
```

커밋 전

```
# 빌드 + 전체 테스트
make -C tests check
dotnet run --project FunLang.Tests
```

CI/CD

```yaml
# .github/workflows/test.yml
- name: Build
  run: dotnet build

- name: Unit Tests
  run: dotnet run --project FunLang.Tests

- name: Integration Tests
  run: make -C tests
```

요약

상황	사용할 도구
CLI 동작 검증	fslit
내부 함수 테스트	Expecto
수학적 성질 검증	FsCheck
회귀 방지	fslit + Expecto
엣지 케이스 발견	FsCheck

관련 문서

- write-fslit-file-tests — fslit 상세 가이드
- setup-expecto-test-project — Expecto 프로젝트 설정
- write-fscheck-property-tests — FsCheck 속성 테스트
- testing-strategies — 전체 테스트 전략