Chapter 1: Foundation & Pipeline

.NET 10과 FsLexYacc를 사용하여 언어 구현의 기초를 설정한다. 이 chapter에서는 아직 실행 가능한 인터프리터를 만들지 않고, Lexer → Parser → AST 파이프라인의 뼈대만 구축한다.

개요

이 chapter에서 다루는 내용: - .NET 10 F# 프로젝트 생성 - FsLexYacc 패키지 설치 - AST 타입 정의 (Discriminated Union) - fslex로 Lexer 작성 - fsyacc로 Parser 작성 - 빌드 순서 설정 (핵심!)

프로젝트 생성

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

# FsLexYacc 패키지 설치
cd FunLang
dotnet add package FsLexYacc --version 11.3.0
```

AST 정의

`Ast.fs` — 언어의 구문 트리를 F# Discriminated Union으로 정의한다.

```
module Ast

/// Expression AST - minimal foundation for lexer/parser pipeline
/// Phase 1: Number only (proof of pipeline)
/// Phase 2 will add: Add, Subtract, Multiply, Divide
type Expr =
    | Number of int
```

Phase 1에서는 숫자만 지원한다. 사칙연산은 Chapter 2에서 추가한다.

Parser 작성

`Parser.fsy` — fsyacc 문법 명세 파일.

```
%{
open Ast
%}

// Token declarations
%token <int> NUMBER
%token EOF

// Start symbol and its type
%start start
%type <Ast.Expr> start

%%
```

```
// Grammar rules
start:
  | NUMBER EOF { Number $1 }
```

구성 요소: - %{ %}: F# 코드 (open 문) - %token: 토큰 선언 (<int>는 토큰이 가지는 값의 타입) - %start, %type: 시작 심볼과 반환 타입 - %% 이후: 문법 규칙 ($1은 첫 번째 심볼의 값)

Lexer 작성

Lexer.fsl — fslex 렉서 명세 파일.

```
{
open System
open FSharp.Text.Lexing
open Parser  // Import token types from generated Parser module

// Helper to get lexeme as string
let lexeme (lexbuf: LexBuffer<_>) =
    LexBuffer<_>.LexemeString lexbuf
}

// Character class definitions
let digit = ['0'-'9']
let whitespace = [' ' '\t']
let newline = ('\n' | '\r' '\n')

// Lexer rules
rule tokenize = parse
  | whitespace+   { tokenize lexbuf }           // Skip whitespace
  | newline       { tokenize lexbuf }           // Skip newlines
  | digit+        { NUMBER (Int32.Parse(lexeme lexbuf)) }  // Integer literal
  | eof           { EOF }                       // End of input
```

핵심 포인트: - open Parser: Parser에서 생성된 토큰 타입(NUMBER, EOF) 사용 - lexeme lexbuf: 현재 매치된 문자열 추출 - tokenize lexbuf: 재귀 호출로 토큰 건너뛰기

빌드 순서 설정 (핵심!)

런타임 순서 ≠ 빌드 순서

- 런타임: Lexer → Parser (문자열을 토큰으로, 토큰을 AST로)
- 빌드: Parser → Lexer (Lexer가 Parser의 토큰 타입을 참조하므로)

FunLang.fsproj:

```
<Project Sdk="Microsoft.NET.Sdk">

  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>net10.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <!-- 1. AST definitions (manually written) -->
```

```
    <Compile Include="Ast.fs" />

    <!-- 2. Parser generator - MUST come before Lexer -->
    <FsYacc Include="Parser.fsy">
      <OtherFlags>--module Parser</OtherFlags>
    </FsYacc>

    <!-- 3. Lexer generator - depends on Parser tokens -->
    <FsLex Include="Lexer.fsl">
      <OtherFlags>--module Lexer --unicode</OtherFlags>
    </FsLex>

    <!-- 4. Generated parser files -->
    <Compile Include="Parser.fsi" />
    <Compile Include="Parser.fs" />

    <!-- 5. Generated lexer file -->
    <Compile Include="Lexer.fs" />

    <!-- 6. Main program -->
    <Compile Include="Program.fs" />
  </ItemGroup>

  <ItemGroup>
    <PackageReference Include="FsLexYacc" Version="11.3.0" />
  </ItemGroup>

</Project>
```

빌드 순서가 틀리면:

```
error FS0039: The namespace or module 'Parser' is not defined
```

Main Program

Program.fs — 파이프라인을 연결하고 테스트한다.

```
open System
open FSharp.Text.Lexing
open Ast

/// Parse a string input and return the AST
let parse (input: string) : Expr =
    let lexbuf = LexBuffer<char>.FromString input
    Parser.start Lexer.tokenize lexbuf

[<EntryPoint>]
let main argv =
    let testInput = "42"

    printfn "FunLang Interpreter - Phase 1: Foundation"
    printfn "=========================================="
    printfn ""
    printfn "Input: %s" testInput
```

```
    try
        let ast = parse testInput
        printfn "AST: %A" ast
        printfn ""
        printfn "Pipeline successful!"
        0
    with
    | ex ->
        printfn "Error: %s" ex.Message
        1
```

Examples

빌드 및 실행

```
$ dotnet build
Build succeeded.

$ dotnet run
FunLang Interpreter - Phase 1: Foundation
========================================

Input: 42
AST: Number 42

Pipeline successful!
```

다른 숫자 테스트

Program.fs에서 testInput을 변경하여 테스트:

```
let testInput = "123"
```

```
$ dotnet run
Input: 123
AST: Number 123

Pipeline successful!
```

빌드 오류 해결

빌드 순서가 잘못되면:

error FS0039: The namespace or module 'Parser' is not defined

해결: .fsproj에서 <FsYacc>가 <FsLex> 위에 있는지 확인.

요약

파일	역할
`Ast.fs`	AST 타입 정의 (Discriminated Union)
`Parser.fsy`	fsyacc 문법 명세 → `Parser.fs`, `Parser.fsi` 생성
`Lexer.fsl`	fslex 렉서 명세 → `Lexer.fs` 생성
`Program.fs`	파이프라인 연결 및 실행
`FunLang.fsproj`	빌드 설정 (순서 중요!)

다음 Chapter

Chapter 2에서는 사칙연산(+, -, *, /)을 추가하고 Evaluator를 구현하여 실제로 계산하는 인터프리터를 만든다.

관련 문서

- setup-fslexyacc-build-order — FsLexYacc 빌드 순서 설정
- write-fslex-lexer — fslex 렉서 작성법
- write-fsyacc-parser — fsyacc 파서 작성법