Chapter 6: Tuples & Pattern Matching

이 장에서는 FunLang에 튜플(Tuple)과 패턴 매칭(Pattern Matching)을 추가한다. 튜플은 고정 크기의 이종 컬렉션으로, 함수에서 여러 값을 반환하거나 관련 데이터를 묶을 때 유용하다.

개요

튜플은 여러 타입의 값을 하나의 복합 값으로 묶는 자료구조다:

- **튜플 생성**: `(1, 2)`, `(1, true, "hello")`
- **패턴 디스트럭처링**: `let (x, y) = (1, 2) in x + y`
- **중첩 튜플**: `((1, 2), 3)`
- **와일드카드 패턴**: `let (_, y) = (1, 2) in y`
- **구조적 동등성**: `(1, 2) = (1, 2)`

AST 확장

Expr 타입에 튜플 표현식 추가

```
// FunLang/Ast.fs

type Expr =
    // ... 기존 케이스들 ...

    // Phase 1 (v3.0): Tuples
    | Tuple of Expr list                // 튜플 표현식: (e1, e2, ...)
    | LetPat of Pattern * Expr * Expr   // 패턴 바인딩: let pat = expr in body
```

케이스	구문	설명
Tuple	(e1, e2, ...)	튜플 표현식 생성
LetPat	let pat = expr in body	패턴 바인딩 let 표현식

Pattern 타입 추가

패턴은 값을 분해하여 변수에 바인딩하는 데 사용된다.

```
/// Pattern for destructuring bindings
and Pattern =
    | VarPat of string          // 변수 패턴: x
    | TuplePat of Pattern list  // 튜플 패턴: (p1, p2, ...)
    | WildcardPat               // 와일드카드 패턴: _
```

패턴	구문	설명
VarPat	x	값을 변수에 바인딩
TuplePat	(p1, p2)	튜플 구조 분해
WildcardPat	_	값을 무시 (바인딩 없음)

Value 타입에 TupleValue 추가

```
and Value =
    | IntValue of int
    | BoolValue of bool
    | FunctionValue of param: string * body: Expr * closure: Env
    | StringValue of string
    | TupleValue of Value list  // 튜플 값
```

TupleValue는 여러 Value를 담는 리스트로 표현된다.

Lexer 확장

새로운 토큰을 추가한다.

```
// FunLang/Lexer.fsl

// Wildcard pattern (반드시 identifier 패턴 전에 와야 함)
| '_'           { UNDERSCORE }

// Comma for tuples
| ','           { COMMA }
```

중요: _ 규칙은 identifier 패턴(ident_start ident_char*)보다 먼저 와야 한다. 그렇지 않으면 _가 identifier로 잘못 인식된다.

토큰 정의

토큰	문자	용도
COMMA	,	튜플 요소 구분자
UNDERSCORE	_	와일드카드 패턴

Parser 확장

토큰 선언

```
// FunLang/Parser.fsy

%token COMMA
%token UNDERSCORE
```

문법 규칙

```
Expr:
    // ... 기존 규칙들 ...

    // 패턴 바인딩 let
    | LET TuplePattern EQUALS Expr IN Expr  { LetPat($2, $4, $6) }

Atom:
```

```
    // ... 기존 규칙들 ...

    // 튜플 표현식: (e1, e2, ...)
    | LPAREN Expr COMMA ExprList RPAREN  { Tuple($2 :: $4) }

// 쉼표로 구분된 표현식 리스트
ExprList:
    | Expr                        { [$1] }
    | Expr COMMA ExprList         { $1 :: $3 }

// 튜플 패턴
TuplePattern:
    | LPAREN PatternList RPAREN   { TuplePat($2) }

PatternList:
    | Pattern COMMA Pattern       { [$1; $3] }
    | Pattern COMMA PatternList   { $1 :: $3 }

Pattern:
    | LPAREN PatternList RPAREN   { TuplePat($2) }
    | IDENT                       { VarPat($1) }
    | UNDERSCORE                  { WildcardPat }
```

문법 구조 설명

튜플 표현식은 `Atom` 레벨에서 파싱된다:

- `(expr)`: 괄호로 묶인 표현식 (기존)
- `(e1, e2, ...)`: 쉼표가 있으면 튜플

패턴 매칭 let은 `Expr` 레벨에서:

- `let x = ...`: 단순 변수 바인딩 (기존)
- `let (p1, p2) = ...`: 튜플 패턴 바인딩

PatternList는 최소 2개 패턴이 필요하다 (단일 요소 튜플은 지원하지 않음).

Evaluator 구현

Tuple: 튜플 평가

```
// FunLang/Eval.fs

| Tuple exprs ->
    let values = List.map (eval env) exprs
    TupleValue values
```

튜플의 각 요소를 순서대로 평가하여 `TupleValue`로 묶는다.

matchPattern: 패턴 매칭

패턴과 값을 매칭하여 바인딩 리스트를 반환한다.

```
/// Match a pattern against a value, returning bindings if successful
let rec matchPattern (pat: Pattern) (value: Value) : (string * Value) list option =
    match pat, value with
    | VarPat name, v -> Some [(name, v)]
    | WildcardPat, _ -> Some []
    | TuplePat pats, TupleValue vals ->
        if List.length pats <> List.length vals then
            None  // Arity mismatch
        else
            let bindings = List.map2 matchPattern pats vals
            if List.forall Option.isSome bindings then
                Some (List.collect Option.get bindings)
            else
                None
    | _ -> None  // Type mismatch
```

핵심 포인트:

- VarPat: 값을 변수에 바인딩
- WildcardPat: 빈 바인딩 반환 (값 무시)
- TuplePat: 요소 개수가 일치해야 하며, 모든 하위 패턴이 매칭되어야 함
- 매칭 실패 시 None 반환

LetPat: 패턴 바인딩 평가

```
| LetPat (pat, bindingExpr, bodyExpr) ->
    let value = eval env bindingExpr
    match matchPattern pat value with
    | Some bindings ->
        let extendedEnv = List.fold (fun e (n, v) -> Map.add n v e) env bindings
        eval extendedEnv bodyExpr
    | None ->
        match pat, value with
        | TuplePat pats, TupleValue vals ->
            failwithf "Pattern match failed: tuple pattern expects %d elements but value has %d"
                      (List.length pats) (List.length vals)
        | TuplePat _, _ ->
            failwith "Pattern match failed: expected tuple value"
        | _ ->
            failwith "Pattern match failed"
```

1. 바인딩 표현식을 평가
2. 패턴 매칭 시도
3. 성공 시 환경 확장 후 본문 평가
4. 실패 시 상세한 에러 메시지 출력

구조적 동등성

튜플은 구조적 동등성(structural equality)을 지원한다.

```
| Equal (left, right) ->
    match eval env left, eval env right with
```

```
    | IntValue l, IntValue r -> BoolValue (l = r)
    | BoolValue l, BoolValue r -> BoolValue (l = r)
    | TupleValue l, TupleValue r -> BoolValue (l = r)  // 구조적 비교
    | _ -> failwith "Type error: = requires operands of same type"
```

F#의 = 연산자는 `Value list`에 대해 자동으로 구조적 비교를 수행한다.

formatValue: 튜플 출력

```
let rec formatValue (v: Value) : string =
    match v with
    | IntValue n -> string n
    | BoolValue b -> if b then "true" else "false"
    | FunctionValue _ -> "<function>"
    | TupleValue values ->
        let formattedElements = List.map formatValue values
        sprintf "(%s)" (String.concat ", " formattedElements)
```

튜플은 `(v1, v2, ...)` 형식으로 출력된다.

Examples

기본 튜플 생성

```
$ dotnet run --project FunLang -- --expr "(1, 2)"
(1, 2)

$ dotnet run --project FunLang -- --expr "(1, true, \"hello\")"
(1, true, "hello")

$ dotnet run --project FunLang -- --expr "(1 + 2, 3 * 4)"
(3, 12)
```

튜플 패턴 디스트럭처링

```
$ dotnet run --project FunLang -- --expr "let (x, y) = (1, 2) in x + y"
3

$ dotnet run --project FunLang -- --expr "let (a, b, c) = (1, 2, 3) in a + b + c"
6

$ dotnet run --project FunLang -- --expr "let (x, y) = (10, 20) in x * y"
200
```

중첩 튜플

```
$ dotnet run --project FunLang -- --expr "((1, 2), 3)"
((1, 2), 3)
```

```
$ dotnet run --project FunLang -- --expr "let ((a, b), c) = ((1, 2), 3) in a + b + c"
6

$ dotnet run --project FunLang -- --expr "(1, (2, 3))"
(1, (2, 3))
```

와일드카드 패턴

```
$ dotnet run --project FunLang -- --expr "let (_, y) = (1, 2) in y"
2

$ dotnet run --project FunLang -- --expr "let (x, _) = (1, 2) in x"
1

$ dotnet run --project FunLang -- --expr "let (_, _, z) = (1, 2, 3) in z"
3

$ dotnet run --project FunLang -- --expr "let (_, (_, z)) = (1, (2, 3)) in z"
3
```

튜플 동등성 비교

```
$ dotnet run --project FunLang -- --expr "(1, 2) = (1, 2)"
true

$ dotnet run --project FunLang -- --expr "(1, 2) = (1, 3)"
false

$ dotnet run --project FunLang -- --expr "(1, 2) <> (1, 3)"
true

$ dotnet run --project FunLang -- --expr "((1, 2), 3) = ((1, 2), 3)"
true
```

함수와 튜플 조합

```
$ dotnet run --project FunLang -- --expr "let swap = fun p -> let (x, y) = p in (y, x) in swa (1, 2)"
(2, 1)

$ dotnet run --project FunLang -- --expr "let fst = fun p -> let (x, _) = p in x in fst (10, 0)"
10

$ dotnet run --project FunLang -- --expr "let snd = fun p -> let (_, y) = p in y in snd (10, 0)"
20
```

REPL 출력 형식

```
$ dotnet run --project FunLang
FunLang REPL - Type an expression and press Enter (Ctrl+C to exit)
> (1, 2, 3)
(1, 2, 3)
> let (x, y) = (10, 20) in x + y
30
> (true, false)
(true, false)
```

AST 확인

```
$ dotnet run --project FunLang -- --emit-ast --expr "(1, 2)"
Tuple [Number 1; Number 2]

$ dotnet run --project FunLang -- --emit-ast --expr "(1, 2, 3)"
Tuple [Number 1; Number 2; Number 3]

$ dotnet run --project FunLang -- --emit-ast --expr "let (x, y) = (1, 2) in x"
LetPat (TuplePat [VarPat "x"; VarPat "y"], Tuple [Number 1; Number 2], Var "x")

$ dotnet run --project FunLang -- --emit-ast --expr "let (_, y) = (1, 2) in y"
LetPat (TuplePat [WildcardPat; VarPat "y"], Tuple [Number 1; Number 2], Var "y")
```

토큰 확인

```
$ dotnet run --project FunLang -- --emit-tokens --expr "(1, 2)"
LPAREN NUMBER(1) COMMA NUMBER(2) RPAREN EOF

$ dotnet run --project FunLang -- --emit-tokens --expr "let (x, _) = (1, 2) in x"
LET LPAREN IDENT(x) COMMA UNDERSCORE RPAREN EQUALS LPAREN NUMBER(1) COMMA NUMBER(2) RPAREN IN IDENT(x) EOF
```

타입 에러

Arity 불일치

```
# 패턴과 튜플의 요소 개수가 다름
$ dotnet run --project FunLang -- --expr "let (x, y) = (1, 2, 3) in x"
Pattern match failed: tuple pattern expects 2 elements but value has 3

$ dotnet run --project FunLang -- --expr "let (x, y, z) = (1, 2) in x"
Pattern match failed: tuple pattern expects 3 elements but value has 2
```

타입 불일치

```
# 튜플 패턴에 비-튜플 값
$ dotnet run --project FunLang -- --expr "let (x, y) = 5 in x"
Pattern match failed: expected tuple value

# 다른 타입 튜플 비교
$ dotnet run --project FunLang -- --expr "(1, 2) = 5"
Type error: = requires operands of same type
```

정리

이 장에서 구현한 내용:

기능	구문	예시
튜플 생성	`(e1, e2, ...)`	`(1, 2, 3)`
변수 패턴	`x`	`let x = 5 in x`
튜플 패턴	`(p1, p2, ...)`	`let (x, y) = (1, 2) in ...`
와일드카드	`_`	`let (_, y) = (1, 2) in y`
중첩 패턴	`((p1, p2), p3)`	`let ((a, b), c) = ... in ...`
구조적 동등성	`=, <>`	`(1, 2) = (1, 2)`

핵심 개념:

- 튜플은 고정 크기 이종 컬렉션이다
- 패턴 매칭으로 튜플을 분해한다
- 와일드카드(_)는 값을 무시한다
- 튜플은 구조적 동등성으로 비교된다

테스트

```
# fslit 테스트
make -C tests

# Expecto 단위 테스트
dotnet run --project FunLang.Tests
```

전체 소스 코드는 FunLang/ 디렉토리를 참조한다.

관련 문서

- write-fsyacc-parser - fsyacc 파서 작성법 (튜플 문법 규칙)
- adapt-tests-for-value-type-evolution - Value 타입 확장 시 테스트 적응
- write-fslex-lexer - fslex 렉서 작성법 (COMMA, UNDERSCORE 토큰)