MiniTS 테스트 케이스 요약

MiniTS 컴파일러 요구사항(어휘 분석 → 파싱/AST → 타입체크 → 코드 생성)을 검증하기 위한 예제들입니다. 모든 테스트는 `examples/*.minits`에 있습니다.

기능 검증 (성공)

T1 — Lexing 커버리지

파일: `examples/01-lexing.minits`

- 실행: `npx minitsc examples/01-lexing.minits -o examples/out/01-lexing.js`
- 목적: 키워드/타입 키워드/식별자/리터럴/연산자/구분자/주석 토큰화 확인.
- 기대 결과: 모든 토큰이 올바른 TokenType으로 생성되고 공백·주석은 무시된다.

토크화된 모습

```
[
  { typeName: "Let", lexeme: "let", line: 5, column: 1 },
  { typeName: "Identifier", lexeme: "alpha", line: 5, column: 5 },
  { typeName: "Colon", lexeme: ":", line: 5, column: 10 },
  { typeName: "NumberType", lexeme: "number", line: 5, column: 12 },
  { typeName: "Equal", lexeme: "=", line: 5, column: 19 },
  { typeName: "NumberLiteral", lexeme: "42", line: 5, column: 21 },
  ...
  { typeName: "Semicolon", lexeme: ";", line: 33, column: 22 },
  { typeName: "RBrace", lexeme: "}", line: 34, column: 1 },
  { typeName: "EOF", lexeme: "", line: 37, column: 1 },
];
```

컴파일 결과

```
let alpha = 42;
let beta = false;
let gamma = "lexing";
let delta = 3.14;
let escaped = 'escape: \\\n \\\t \\\\\"quote\\\\\"';
function inc(v) {
  return v + 1;
}
if (alpha >= 40 && !beta) {
  alpha = (alpha + 1) * 2 - 5 / 5;
} else {
  alpha = alpha - 1;
```

```
}
while (alpha < 50) {
  alpha = inc(alpha);
}
{
  let i = 0;
  while (i < 3) {
    gamma = gamma + "!";
    i = i + 1;
  }
}
```

T2 — Parsing + AST 통합

파일: `examples/02-parsing.minits`

- 실행: `npx minitsc examples/02-parsing.minits -o examples/out/02-parsing.js`
- 목적: 우선순위/좌결합 파싱과 if/while/for/함수 선언·호출까지 AST 생성 확인.
- 기대 결과: `combineAndWalk` 내부 블록·대입·이항/단항/호출 노드가 문법에 맞게 생성된다.
- 컴파일 결과: `examples/out/02-parsing.js`

AST 로그(요약)


```
{
  "kind": "Program",
  "body": [
    {
      "kind": "VarDecl",
      "name": "a",
      "varType": "number",
      "init": {
        "kind": "BinaryExpr",
        "operator": "+",
        "left": { "kind": "NumberLiteralExpr", "value": 1 },
        "right": {
          "kind": "BinaryExpr",
          "operator": "*",
          "left": { "kind": "NumberLiteralExpr", "value": 2 },
          "right": { "kind": "NumberLiteralExpr", "value": 3 }
        }
      }
    },
...
    {
      "kind": "VarDecl",
      "name": "e",
      "varType": "boolean",
      "init": {
        "kind": "BinaryExpr",
```

```
        "operator": "||",
        "left": {
          "kind": "BinaryExpr",
          "operator": "&&",
          "left": {
            "kind": "BinaryExpr",
            "operator": "<",
            "left": {
              "kind": "BinaryExpr",
              "operator": "+",
              "left": { "kind": "NumberLiteralExpr", "value": 1 },
              "right": {
                "kind": "BinaryExpr",
                "operator": "*",
                "left": { "kind": "NumberLiteralExpr", "value": 2 },
                "right": { "kind": "NumberLiteralExpr", "value": 3 }
              }
            },
            "right": { "kind": "NumberLiteralExpr", "value": 10 }
          },
          "right": {
            "kind": "BinaryExpr",
            "operator": "==",
            "left": { "kind": "NumberLiteralExpr", "value": 4 },
            "right": { "kind": "NumberLiteralExpr", "value": 4 }
          }
        },
        "right": { "kind": "BooleanLiteralExpr", "value": false }
      }
    },
...
    {
      "kind": "VarDecl",
      "name": "result",
      "varType": "number",
      "init": {
        "kind": "CallExpr",
        "callee": { "kind": "IdentifierExpr", "name": "combineAndWalk" },
        "args": []
      }
    }
  ]
}
```


컴파일 결과

```
let a = 1 + 2 * 3;
let b = (1 + 2) * 3;
let c = 10 - 3 - 2;
let d = (10 / 2) * 3;
let e = (1 + 2 * 3 < 10 && 4 == 4) || false;
let f = !(a < b) || a === b;
```

```
function combineAndWalk() {
  let counter = 0;
  if (a < b && e) {
    counter = counter + 1;
  } else {
    counter = counter - 1;
  }
  while (counter < 3) {
    counter = counter + 2;
  }
  {
    let i = 0;
    while (i < 2) {
      counter = counter + i;
      i = i + 1;
    }
  }
  let folded = (a + b) / (c - 2);
  return folded + counter + d;
}
let result = combineAndWalk();
```

T3 — 타입 규칙 성공 경로

파일: `examples/03-type-rules-success.minits`

- 실행: `npx minitsc examples/03-type-rules-success.minits -o examples/out/03-type-rules-success.js`
- 목적: 타입 규칙 요약을 위반 없이 통과하는 정상 조합 검증(암묵 변환 없음).
- 기대 결과: 숫자/문자열 연산, 비교·논리, 함수 호출/반환 타입이 모두 통과하며 타입 오류가 없다.

컴파일 결과

```
let n1 = 4 + 5;
let n2 = n1 - 3;
let n3 = n2 * 2;
let n4 = n3 / 3;
let s1 = "mini";
let s2 = "ts";
let s3 = s1 + s2;
let cmp = n1 > n2;
let logic = (cmp && true) || !false;
function andAll(a, b) {
  return a && b;
}
function echoStr(name) {
  return "Hello " + name;
}
let ok = andAll(cmp, logic);
let greet = echoStr(s3);
```

T4 — 변수 선언/대입

파일: `examples/04-var-decl-assign.minits`

- 실행: `npx minitsc examples/04-var-decl-assign.minits -o examples/out/04-var-decl-assign.js`
- 목적: 타입 명시/추론, 선언 후 대입, 동일 타입 유지 확인.
- 기대 결과: 심볼 테이블에 기록되고 타입 일관성이 유지된다.
- 컴파일 결과: `examples/out/04-var-decl-assign.js`

컴파일 결과

```
let a = 1;
let b = 2;
let c;
c = a + b;
let flag = true;
flag = false;
let msg = "hello";
msg = "world";
let total = c + 7;
let switched = total;
switched = switched - 1;
```

T5 — 산술·비교·논리 연산

파일: `examples/05-arith-compare-logical.minits`

- 실행: `npx minitsc examples/05-arith-compare-logical.minits -o examples/out/05-arith-compare-logical.js`
- 목적: 산술/비교/논리 연산과 문자열 concat 파싱·타입체크.
- 기대 결과: 올바른 타입 조합은 통과, 잘못된 조합 시 타입 오류가 발생해야 함.
- 컴파일 결과: `examples/out/05-arith-compare-logical.js`

컴파일 결과

```
let x = 10;
let y = 3;
let add = x + y;
let sub = x - y;
let mul = x * y;
let div = x / y;
let lt = x < y;
```

```
let le = x <= y;
let gt = x > y;
let ge = x >= y;
let eq = x == y;
let seq = x === y;
let b1 = true;
let b2 = false;
let andResult = b1 && b2;
let orResult = b1 || b2;
let notResult = !b2;
let s1 = "hello";
let s2 = " world";
let s3 = s1 + s2;
let complex = add > mul || (div == 3 && !lt);
```

T6 — 조건문(if / if-else)

파일: `examples/06-if-else.minits`

- 실행: `npx minitsc examples/06-if-else.minits -o examples/out/06-if-else.js`
- 목적: 다중 분기와 boolean 조건 강제 확인.
- 기대 결과: 조건이 boolean일 때만 통과하며, 상태 갱신이 AST/타입체크를 통과한다.
- 컴파일 결과: `examples/out/06-if-else.js`

컴파일 결과

```
let score = 85;
let grade = "F";
if (score >= 90) {
  grade = "A";
} else {
  if (score >= 80) {
    grade = "B";
  } else {
    if (score >= 70) {
      grade = "C";
    } else {
      grade = "D";
    }
  }
}
let passed = false;
if (score >= 60) {
  passed = true;
}
let bonus = 5;
if (passed && grade == "B") {
  score = score + bonus;
}
```

T7 — while 반복문

파일: `examples/07-while.minits`

- 실행: `npx minitsc examples/07-while.minits -o examples/out/07-while.js`
- 목적: while 파싱·타입체크와 블록 스코프 확인.
- 기대 결과: 조건은 boolean, 내부 대입이 타입 일관성을 유지한다.
- 컴파일 결과: `examples/out/07-while.js`

컴파일 결과

```
let i = 1;
let sum = 0;
while (i <= 5) {
  sum = sum + i;
  i = i + 1;
  if (sum > 6) {
    let over = true;
    if (over) {
      sum = sum + 0;
    }
  }
}
```

T8 — for 반복문(디소거)

파일: `examples/08-for.minits`

- 실행: `npx minitsc examples/08-for.minits -o examples/out/08-for.js`
- 목적: for 헤더 파싱과 while 블록 디소거 검증.
- 기대 결과: init/cond/step 형태가 while+블록으로 변환되고 타입 오류 없이 통과한다.
- 컴파일 결과: `examples/out/08-for.js`

컴파일 결과

```
function sumTo(n) {
  let acc = 0;
  {
    let i = 0;
    while (i < n) {
      acc = acc + i;
      i = i + 1;
    }
```

```
  }
  let j = 0;
  while (j < 3) {
    acc = acc + j;
    j = j + 1;
  }
  return acc;
}
let result = sumTo(5);
```

T9 — 함수 선언 & 호출

파일: `examples/09-functions.minits`

- 실행: `npx minitsc examples/09-functions.minits -o examples/out/09-functions.js`
- 목적: 여러 시그니처, 중첩 호출, 인자·반환 타입 검증.
- 기대 결과: 시그니처가 기록되고 호출 시 타입/개수 일치 검사가 통과한다.
- 컴파일 결과: `examples/out/09-functions.js`

컴파일 결과

```
function add(a, b) {
  return a + b;
}
function isPositive(x) {
  return x > 0;
}
function greet(name) {
  return "Hello, " + name;
}
let s = add(3, 4);
let ok = isPositive(s);
let message = greet("MiniTS");
function doubleAdd(x) {
  return add(x, add(1, 2));
}
let d = doubleAdd(5);
```

T10 — 전체 통합

파일: `examples/10-full-program.minits`

- 실행: `npx minitsc examples/10-full-program.minits -o examples/out/10-full-program.js`
- 목적: 변수/연산/조건/반복/함수/문자열 concat 전체 흐름 통합.
- 기대 결과: 파싱→AST→타입체크→코드생성까지 에러 없이 완료된다.

- 컴파일 결과: `examples/out/10-full-program.js`

컴파일 결과

```
function sumTo(n) {
  let acc = 0;
  {
    let i = 0;
    while (i < n) {
      acc = acc + i;
      i = i + 1;
    }
  }
  return acc;
}
function factorial(n) {
  let result = 1;
  let i = 1;
  while (i <= n) {
    result = result * i;
    i = i + 1;
  }
  return result;
}
let limit = 5;
let totalSum = sumTo(limit);
let fact = factorial(5);
let ok = totalSum < fact && fact === 120;
let status = "unknown";
if (ok) {
  status = "OK";
} else {
  status = "NG";
}
let tags = "";
{
  let t = 0;
  while (t < 3) {
    tags = tags + "#";
    t = t + 1;
  }
}
let counter = 0;
while (counter < 3) {
  counter = counter + 1;
}
```

의도된 오류 검증

T11 — Lexer 오류

파일: `examples/11-lexer-error.minits`

- 실행: `npx minitsc examples/11-lexer-error.minits -o examples/out/11-lexer-error.js`
- 목적: 알 수 없는 문자 처리.
- 기대 결과: `Unexpected character ...`로 중단, 파서/타입체커로 진행되지 않음.
- 컴파일 결과: 생성되지 않음(의도된 오류)

컴파일 결과 (에러 로그)

```
Unexpected character '@' at line 5, column 1
```

T12 — Parse 오류

파일: `examples/12-parse-error.minits`

- 실행: `npx minitsc examples/12-parse-error.minits -o examples/out/12-parse-error.js`
- 목적: 세미콜론 누락 등 구문 오류 감지.
- 기대 결과: `[ParseError] Expected ';' ...` 메시지 출력.
- 컴파일 결과: 생성되지 않음(의도된 오류)

컴파일 결과 (에러 로그)

```
[ParseError] Expected ';' after variable declaration at line 5, column 1.
Got: Let (let)
```

T13 — Type 오류(대입)

파일: `examples/13-type-error-assign.minits`

- 실행: `npx minitsc examples/13-type-error-assign.minits -o examples/out/13-type-error-assign.js`
- 목적: 잘못된 대입 타입 검출.
- 기대 결과: `[TypeError] Cannot assign 'string' to variable 'x' of type 'number'`.
- 컴파일 결과: 생성되지 않음(의도된 오류)

컴파일 결과 (에러 로그)

```
[TypeError] Cannot assign 'string' to variable 'x' of type 'number'
```

T14 — Type 오류(함수 호출)

파일: `examples/14-type-error-call.minits`

- 실행: `npx minitsc examples/14-type-error-call.minits -o examples/out/14-type-error-call.js`
- 목적: 함수 인자 타입 불일치 감지.
- 기대 결과: `[TypeError] Argument 2 of 'add' expects 'number', got 'string'`.
- 컴파일 결과: 생성되지 않음(의도된 오류)

컴파일 결과 (에러 로그)

```
[TypeError] Argument 2 of 'add' expects 'number', got 'string'
```

T15 — 암묵 변환 금지

파일: `examples/15-no-implicit-cast.minits`

- 실행: `npx minitsc examples/15-no-implicit-cast.minits -o examples/out/15-no-implicit-cast.js`
- 목적: 자동 캐스팅이 없음을 검증(number+string 등).
- 기대 결과: 여러 `[TypeError]`가 발생해 컴파일이 중단된다.
- 컴파일 결과: 생성되지 않음(의도된 오류)

컴파일 결과 (에러 로그)

```
[TypeError] Operator '+' requires (number, number) or (string, string), got 'number' and 'string'
```

실행 로그 예시

`npm run examples` (scripts/run-examples.mjs) 실행 시 성공/실패를 요약한다. 오류 케이스는 의도된 실패이다.

```
=== Running all MiniTS example programs ===
... (성공) 01~10 컴파일 완료 ...
--- Compiling 11-lexer-error.minits ---
```

```
Unexpected character '@' at line 5, column 1
ERROR while compiling 11-lexer-error.minits

--- Compiling 12-parse-error.minits ---
[ParseError] Expected ';' after variable declaration at line 5, column 1.
Got: Let (let)
ERROR while compiling 12-parse-error.minits

--- Compiling 13-type-error-assign.minits ---
[TypeError] Cannot assign 'string' to variable 'x' of type 'number'
ERROR while compiling 13-type-error-assign.minits

--- Compiling 14-type-error-call.minits ---
[TypeError] Argument 2 of 'add' expects 'number', got 'string'
ERROR while compiling 14-type-error-call.minits

--- Compiling 15-no-implicit-cast.minits ---
[TypeError] Operator '+' requires (number, number) or (string, string),
got 'number' and 'string'
ERROR while compiling 15-no-implicit-cast.minits
```