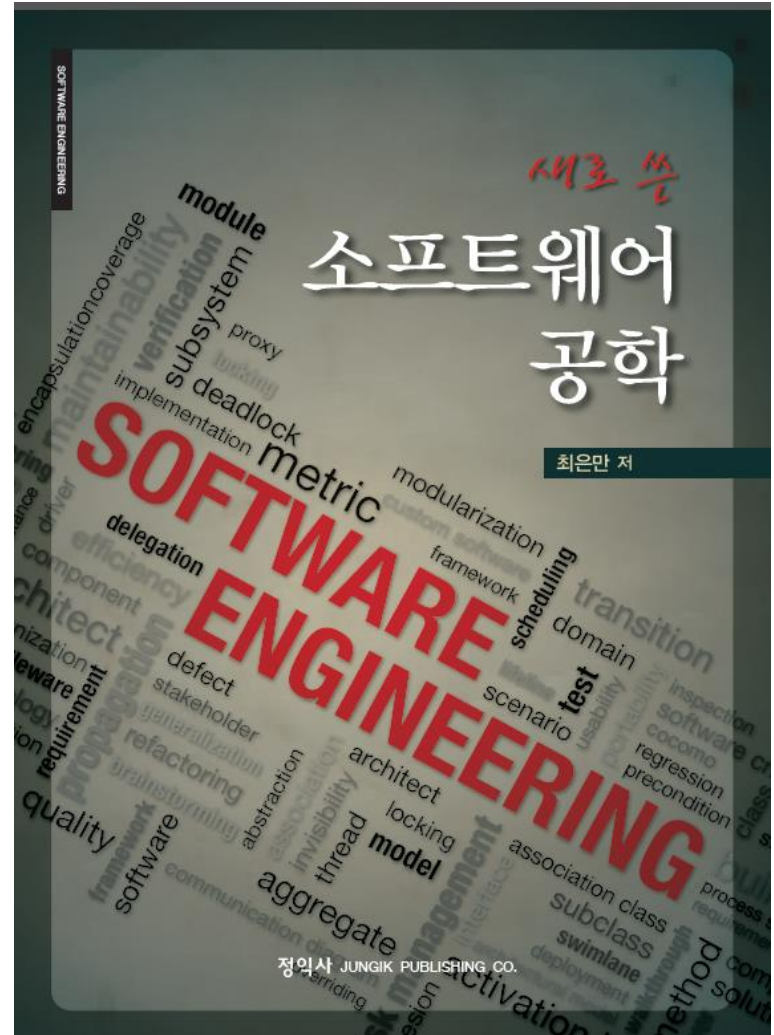


# Lecture #9: 테스트

## 6차 개정판



# 학습 목표

- 테스트 기초와 원리
- 블랙박스 테스트
- 화이트 박스 테스트
- 객체지향 테스트
- 통합 테스트
- 시스템 및 인수 테스트
- 테스트 자동화 도구

# 9.1 테스트 기초

- 소프트웨어의 **정확성**을 입증하는 과정
  - 결함이나 원치 않는 동작을 찾는 것
  - 요구와 제약에 맞는지 검증
- 좋은 테스트란 숨어있는 오류를 잘 발견하는 것

verification(검토, 검증) 과 validation(확인) 의 해결책:  
inspection, testing,



# 테스팅 용어

- 오류(error)
  - 프로그램 실행 결과가 예상결과와 다른 경우
  - 결함 및 고장을 일으키게 한 **인간의 실수**
- 결함(fault) : **오류가 결과물에 구체화된 것**
  - 버그(bug)
  - 소프트웨어 오작동의 원인
  - ➔ default
- 고장(failure)
  - 명세로 작성된 요구와 기능을 제대로 수행할 수 없는 경우
  - 모든 결함이 고장을 발생하는 것은 아님

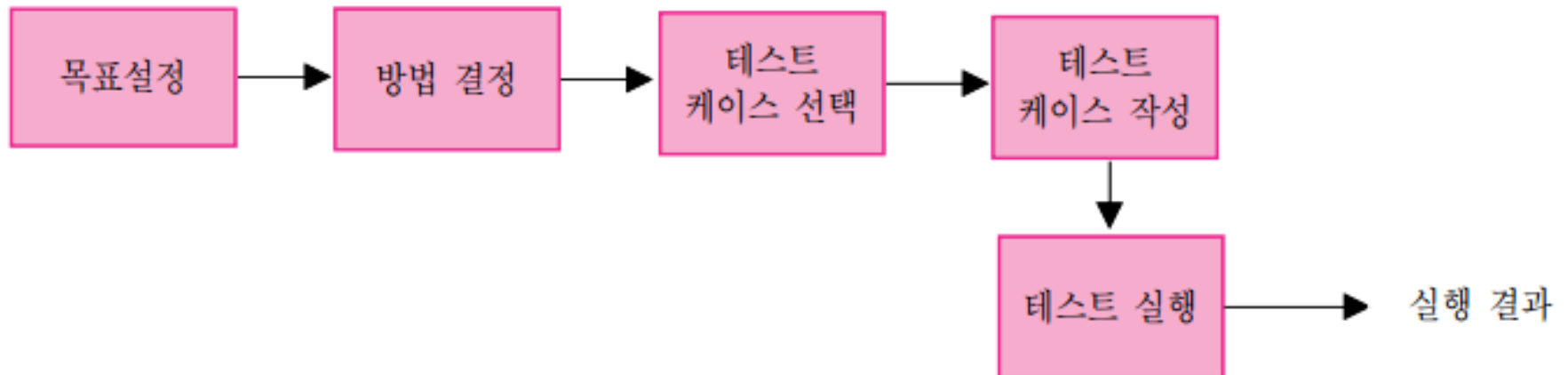
**원인(error, fault) ==> 현상 (failure)**

# 테스팅 원리

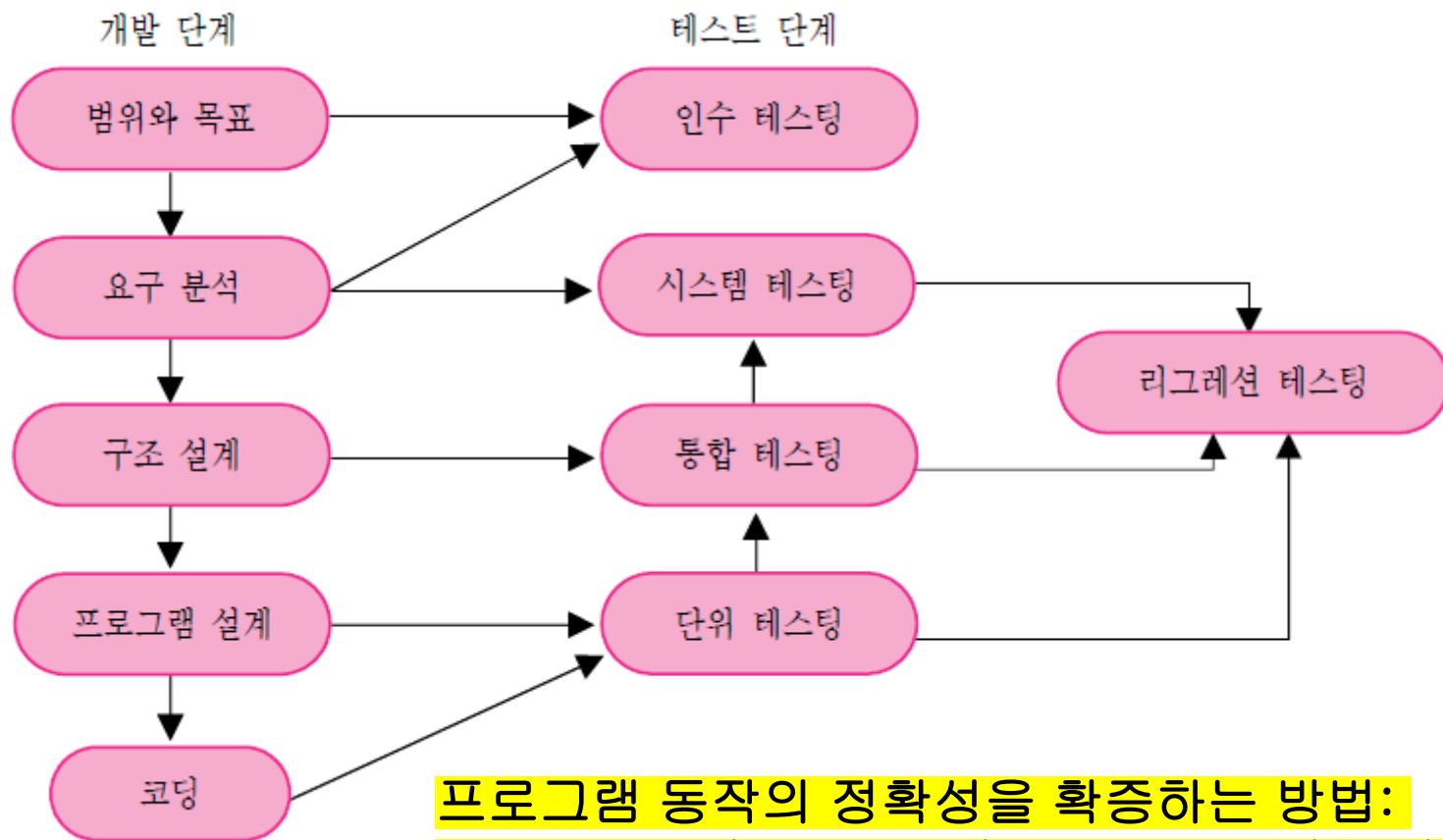
- 테스팅은 오류를 발견하려고 프로그램을 실행시키는 것
- 완벽한 테스팅을 불가능
- 테스팅은 창조적이면서 어려운 작업
- 테스팅은 오류의 유입을 방지
- 테스팅은 구현과 관계없는 독립된 팀에 의해 수행되어야함



# 테스팅 과정



# 개발 단계별 테스트



프로그램 동작의 정확성을 입증하는 방법:  
Verification (검증, 검토)와 Validation(확인)을 위  
한 방법: testing, inspection

# 테스트 케이스

- 시험 대상 단위 별로 묶음

고유번호	테스트 대상	테스트 조건	테스트 데이터	예상 결과
FT-1-1	로그인 기능	시스템 초기 화면	정상적인 사용자 ID('gdhong')와 비밀번호('1234')	시스템 입장
FT-1-2	"	"	비정상적 사용자 ID('%\$##')와 비밀번호(' ')	로그인 오류 메시지

**테스트 시나리오:** 테스트 케이스를 적용하는 순서에 따라 여러 개의 테스트 케이스를 묶은 집합



## 9.2 블랙박스 테스트

- 프로그램 구조를 고려하지 않음
- 테스트 케이스는 프로그램이나 모듈의 요구나 명세를 기초로 결정
- 입력과 출력에 대해 알아야 함
- 기능테스트(functional testing)
- 가능한 모든 기능을 전부 테스트 하는 것이 좋음



# 동치 클래스

- 프로그램에서 같은 부분을 구동시키고 결과를 확인하는 값들의 대푯값
  - 정상적인 동치클래스
  - 비정상적인 동치클래스
- 모든 대표 값을 찾아내어 각 클래스에서 하나의 값으로 프로그램을 실행시킨다면 전수 테스트와 같음

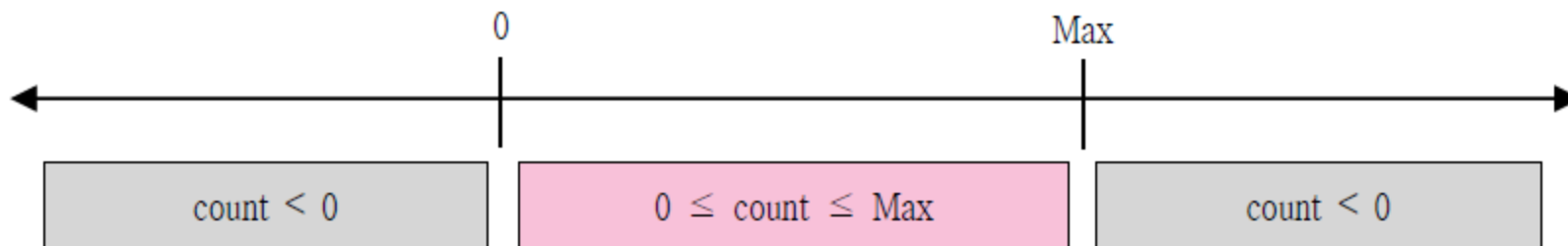
– 동작이나 출력이 달라지도록 하는 테스트 케이스 그룹

– 예) 컴공대학원 입학전형 기준: 학점  $\geq 3.0$  여부, 전공이 컴공 여부  $\rightarrow$  4개의 동치 클래스

<예> 입력 범위 조건  $0 \leq \text{count} \leq \text{Max}$  의 동치 클래스

정상  $0 \leq \text{count} \leq \text{Max}$

비정상  $\text{count} < 0, \text{count} > \text{Max}$



# 경계값 분석

- 동치 클래스의 경계에서 문제를 발생하는 특수한 값이 존재
- 동치 클래스 경계에 있는 값을 가진 테스트 케이스는 높은 효율을 가짐
- 동치 클래스의 경계에 있는 값을 테스트 입력으로 선택

<예> 하나의 입력 값  $X$ 에 대한 테스트 케이스.

예를 들면,  $X$ 는 정수 값을 가짐. 입력값 조건:  $\min \leq X \leq \max$   
케이스:  $\min-1, \min, \min+1, \max-1, \max, \max+1$



# 경계값 분석

<예> 두개의 입력 값 X, Y에 대한 테스트 케이스

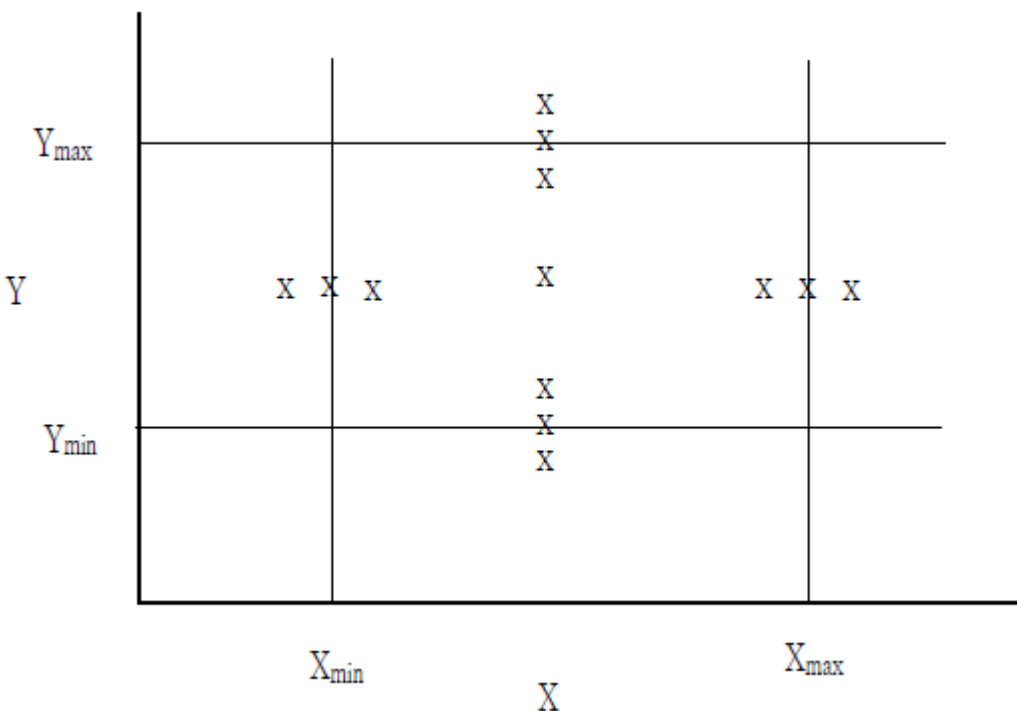
입력값 조건:  $X_{min} \leq X \leq X_{max}$

$Y_{min} \leq Y \leq Y_{max}$

예) 정수값 X의 경계값:

$X_{min}-1, X_{min}, X_{min}+1,$   
 $X_{max}-1, X_{max}, X_{max}+1$

정수값 Y의 경계값: X  
와 같은 수의 케이스 존재



n개 변수인 경우의 테스트 케이스 숫자:

$\rightarrow 6^n$

$\rightarrow 6n + 1$  : 한 변수는 경계값을, 다른 변수들은 정상값으로 설정하는 케이스 숫자 + 1 (모든 변수에 대해 어떤 정상값으로 설정한 경우)

# 원인과 결과 그래프(1)

- 입력 조건(→ 원인)의 조합을 체계적으로 선택하는 테스트 기법

- 노드와 기호로 표시

- 노드: 원인(입력조건), 결과(출력 조건)
- 기호:  $\wedge$  (and),  $\vee$  (or),  $\sim$  (not)

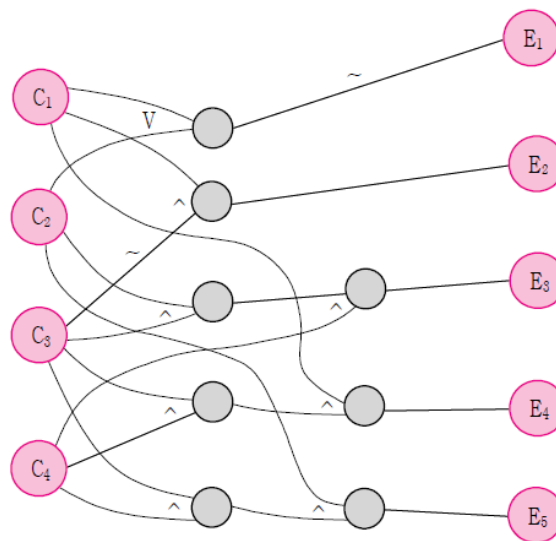
<예>

원인:

- c1. 명령어가 입금
- c2. 명령어가 출금
- c3. 계정 번호가 정상
- c4. 트랜잭션 금액이 정상

결과:

- e1. '명령어 오류'라고 인쇄
- e2. '계정 번호 오류'라고 인쇄
- e3. '출금액 오류'라고 인쇄
- e4. 트랜잭션 금액 출금
- e5. 트랜잭션 금액 입금



# 원인과 결과 그래프(2)

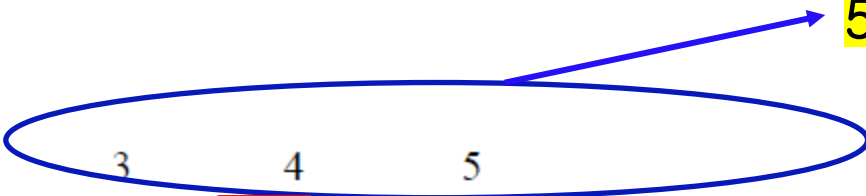
- 각각의 결과들에 대하여 **조건의 조합**을 나열

<예>

x: don't care

0: false

1: true

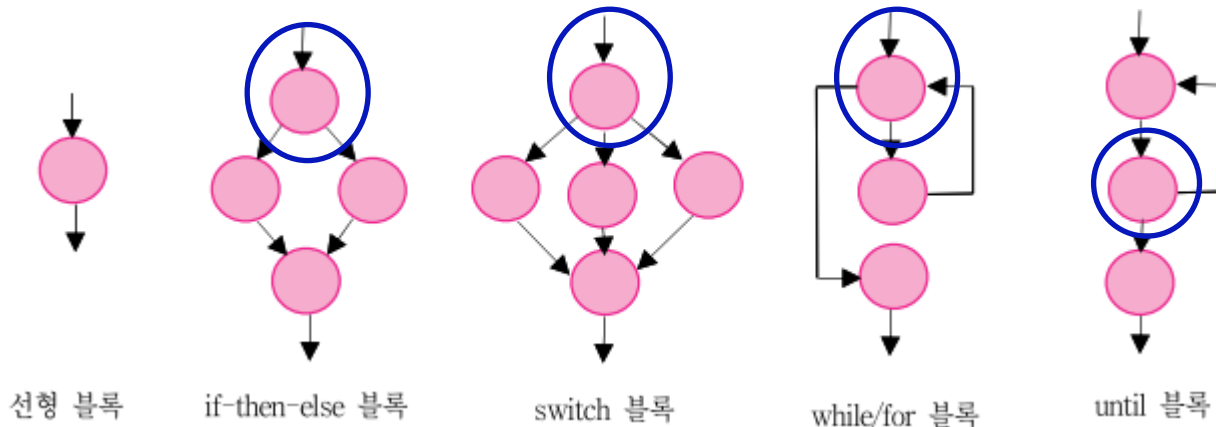


No.	1	2	3	4	5
c1	0	1	x	x	1
c2	0	x	1	1	x
c3	x	0	1	1	1
c4	x	x	0	1	1
e1	1				
e2		1			
e3			1		
e4				1	
e5					1

5 개의 조합

## 9.3 화이트박스 테스트

- 모듈의 논리적인 구조를 체계적으로 점검하는 구조적 테스트
- 여러가지 프로그램 구조를 기반으로 테스트
- 논리 흐름도(logic-flow diagram)을 이용
  - 노드: 모듈내의 모든 세그먼트
  - 간선: 제어 흐름



# 기본 경로 테스트

- 기본경로(basis path)

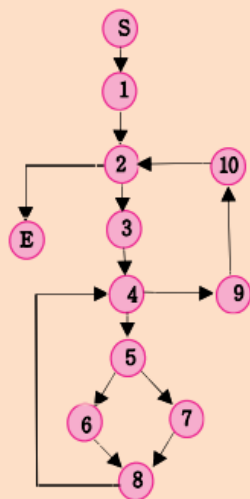
- 독립적인 논리 흐름을 검사하는 테스트 케이스를 생성

- 기본 경로: 시작 노드에서 종료 노드까지의 서로 독립된 경로로써 사이클은 최대 한번만 지나야 함.

(Independent Paths : An independent path in the control flow graph is the one which introduces at least one new edge that has not been traversed before the path is defined)

<예> Remove (중복제거) 함수에 대한 논리흐름 그래프와 테스트 케이스

```
public void Remove(LinkedList list) {  
    Item p, q;  
    ① p = lItem(list.getFirst());  
    ② while(p!=null) {  
    ③     q=(Item)p.getNext();  
    ④     while (q!=null) {  
    ⑤         if(p.compareTo(q)==0)  
    ⑥             list.remove(q);  
    ⑦         else q=(Item)p.getNext();  
    ⑧     }  
    ⑨     p=(Item)p.getNext();  
    ⑩ }  
}
```



기본경로 1. S-1-2-E: 빈 리스트

기본경로 2. S-1-2-3-4-9-10-2-E: 한 개의 요소를 가진 리스트

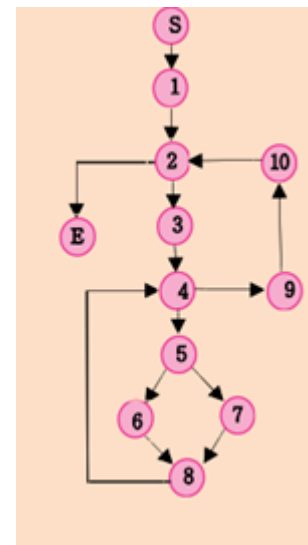
기본경로 3. S-1-2-3-4-5-6-8-4-9-10-2-E: 중복 요소를 가진 리스트

기본경로 4. S-1-2-3-4-5-7-8-4-9-10-2-E: 중복 요소가 없는 리스트



# 싸이클로매틱(순환) 복잡도

- 기본 경로의 수를 결정하는 이론
- 싸이클로매틱 복잡도 계산 3가지 방법
  - 폐쇄 영역의 수 +1 : 논리 흐름 그래프는 이차원 평면을 여러 영역으로 나누며, 이 중 폐쇄된 영역의 수에 1을 더한 값
  - 노드와 간선의 수 : 간선의 수 (앞의 예: 14)에서 노드의 수(앞의 예: 12)를 빼고 2를 더한 값
  - 단일 조건의 수 +1 : 참과 거짓으로 판별되는 원자적 조건의 수(앞의 예: 3개, 즉 2번, 4번과 5번 노드)에 1을 더한 값
    - \* 단일조건: AND, or 로 연결되지 않은 단일조건
- 3가지 방법의 값이 같아야 함



# 테스트 커버리지

- 테스트를 어느 정도 완벽히 수행할 것인가의 **기준**

- **노드 커버리지**

- 논리 흐름 그래프의 각 노드가 테스트 케이스에 의하여 적어도 한 번씩 방문되어야 하는 검증기준
- 프로그램 문장 100% 커버

- **간선 커버리지**

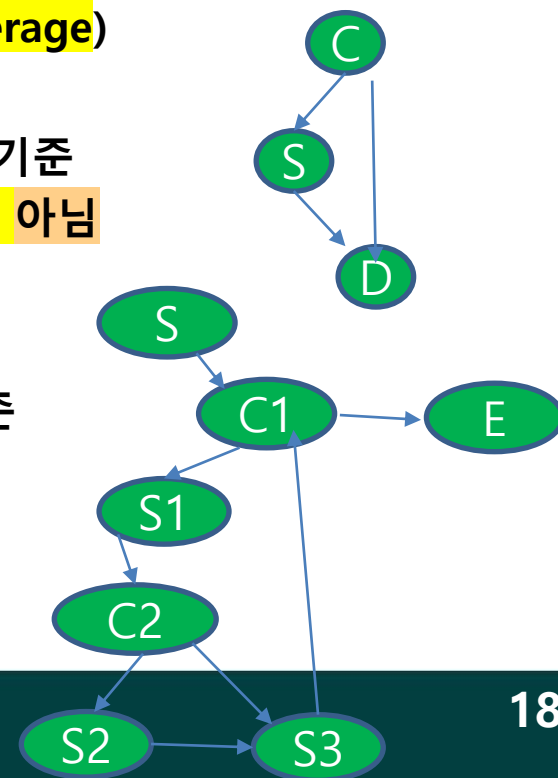
- 논리 흐름 그래프의 각 간선이 테스트 케이스에 의하여 적어도 한 번씩 방문되어야 하는 검증기준
- 모든 분기점 테스트(**Branch coverage or decision coverage**)

- **기본 경로 커버리지**

- 모든 기본 경로가 적어도 한 번씩 방문되어야 하는 검증기준
- “**간선 커버리지 100% → 기본 경로 커버리지 100%**” 는 아님
- 

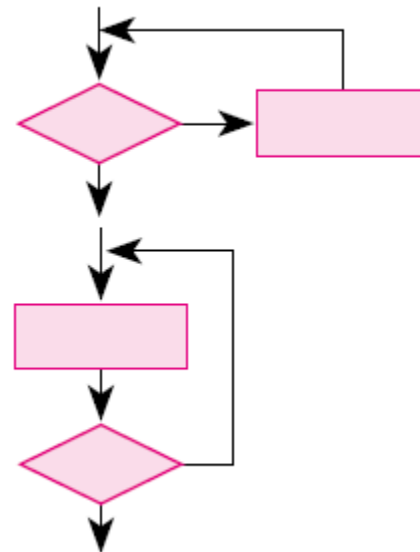
- **모든 경로 커버리지**

- 모든 가능한 경로를 적어도 한 번씩 테스트하는 검증기준
- 현실적으로 불가능 ← 반복문 고려시



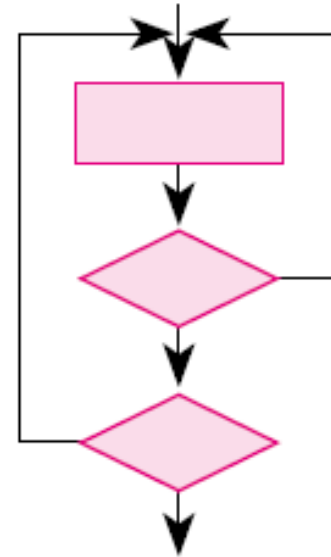
# 반복문의 테스트(1)

- Beizer 반복구조 분류
- 단순 반복
- 경계값 분석 방법 이용
  - 반복 구조를 들어가지 않고 생략
  - 반복 구조 안에서 한 번 반복
  - 반복 구조 안에서 두 번 반복
  - 일정한 횟수의 반복
  - 반복 최대 횟수 - 1 만큼 반복
  - 반복 최대 횟수만큼 반복
  - 반복 최대 횟수 + 1 만큼 반복



# 반복문의 테스트(2)

- 중첩된 반복
- 가장 내부에 있는 반복 구조부터 테스트 (단, 외부 반복 구조는 최소 반복횟수로 지정)
  - 최소 횟수의 반복
  - 최소 횟수보다 하나 많은 반복
  - 범위 내 임의의 횟수 반복
  - 최대 횟수보다 하나 적은 반복
  - 최대 횟수의 반복
  - 외부로 향하여 다음 반복구조를 테스트



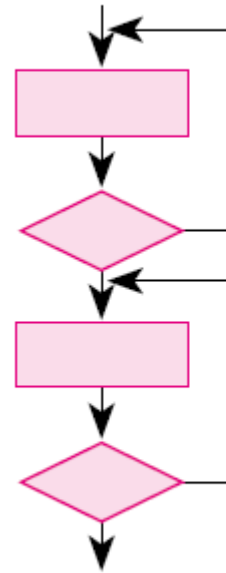
# 반복문의 테스트(3)

## ● 연속된 반복

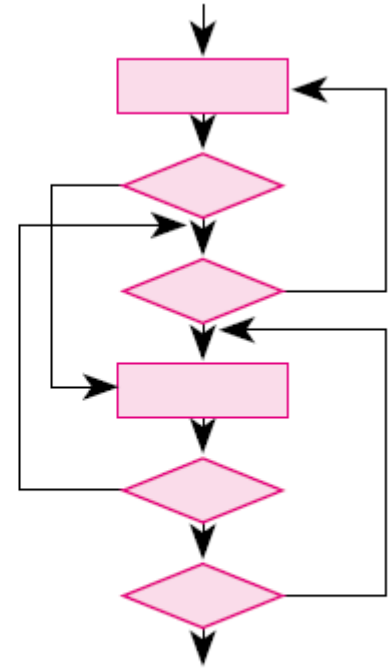
- 반복구조가 서로 독립적이면 '단순반복'
- 반복구조가 어느 한쪽의 제어변수가 다른 한쪽의 제어변수에 의존적인 관계라면 '중첩된 반복'

## ● 비구조화 반복

- 구조적 반복 형태로 변경하여 테스트



(c) 연속된 반복



(d) 비구조화 반복

## 9.4 객체지향 테스트

- 객체지향 방식의 프로그램에 적용
- 사용사례 기반 테스트
- 상태 기반 테스트

