11/22

과제2
유스케이스에대한 시나리오 → 테스트케이스작성 (명세서)
↳ 올라오다이어그램으로!
원래는과제1도...

10장. 화이트박스 테스트

숙명여자대학교
SOOKMYUNG WOMEN'S UNIVERSITY

Contents

1. 테스트 개요 (1/5)

- 소프트웨어 품질 ≈ 결함 예방 ⇒ SW공학 기법 적용
- 유입된 결함 제거를 위해 테스팅은 필수적
- SW 테스트 : SW의 정확성을 확증하는 과정
  - 소프트웨어 내에 존재하는 오류 발견
  - 소프트웨어 요구사항에 충족하는지 확인
  - 소프트웨어 명세에 충족하는지 확인
  - 소프트웨어 출시 이후 발생할 수 있는 결함을 예방
  - 개발된 소프트웨어에 신뢰성을 높여 주기 위한 작업

1. 테스트 개요 (2/5)

- 소프트웨어 테스트 활동
  - 대상 소프트웨어를 테스트하기 위한 입력 데이터 준비
  - 입력 데이터를 이용하여 소프트웨어를 실행하고 결과를 모니터링
  - 의도된 결과와 실제 실행 결과 비교
  - 테스팅은 구현과 관계없는 독립된 팀에 의해 수행되어야 함
- 테스트 결과에서 결함이 발견되지 않았다 하더라도 이는 결함이 없음을 의미하는 것은 아님

테스팅기법에 따라

→ 결함을 많이 찾아낼수록 좋은 테스트

1. 테스트 개요 (3/5)

- 입력 데이터
  - 정상, 비정상적인 데이터 포함, 데이터 양 고려
  - 모든 경우의 수에 대해 테스트가 불가능하므로 높은 확률로 오류를 찾아 낼 수 있도록 좋은 테스트 데이터가 필요함
  - 예)

> - 주어진 문제: 2개의 정수형 입력 X와 Y를 비교하는 모듈을 테스트한다.
> - 가정 사항
>   - 워드 사이즈는 32비트로 가정한다.
>   - 처리기CPU가 두 입력에 대한 비교 연산 하나를 수행하는 데 0.1μs가 소요된다.

    - 변수 X와 Y의 모든 가능한 입력의 조합 = 각각 2^{32} 즉, 2×2^{32}
    - 완전 자동화로 테스트 수행해도 58,494년 필요

1. 테스트 개요 (4/5)

- 테스팅 용어
  - 오류(Error) :
    - 프로그램 실행결과가 예상결과와 다른 경우
    - 결함 및 고장을 일으키게 한 인간의 실수
  - 결함(Fault, Defect)
    - 버그(bug)
    - 소프트웨어 고장(오작동)의 원인
  - 고장(failure)
    - 명세로 작성된 요구와 기능을 제대로 수행할 수 없는 경우가 외부에서 관찰되는 상황
    - 모든 결함이 고장을 발생하는 것은 아님

1. 테스트 개요 (5/5)

- 예) 다음과 같은 요구사항을 고려해보자

> 속도를 출력한다. 속도(S)는 거리(d)를 시간(t)으로 나누어 구한다.

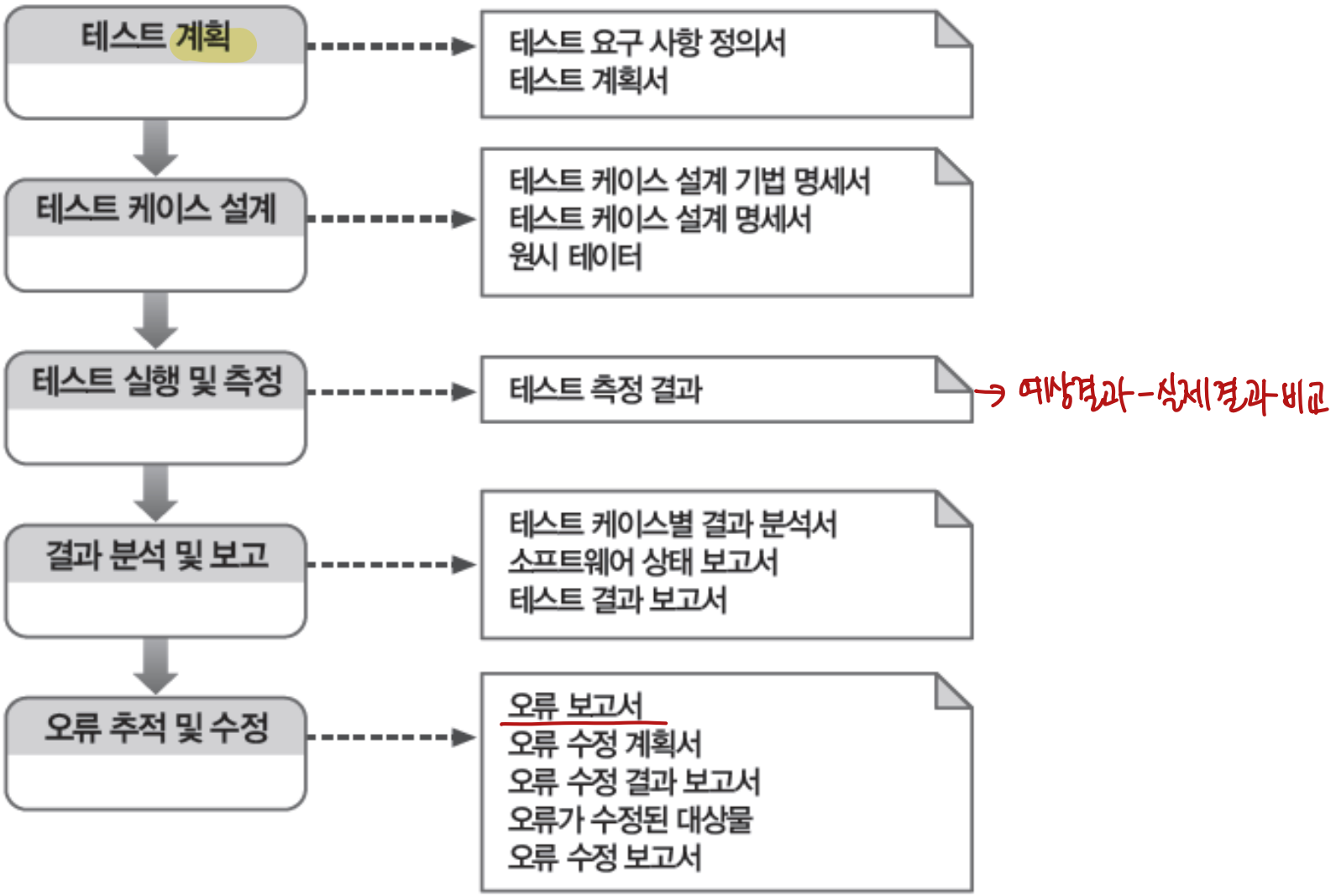
- 다음과 같이 코딩을 할 수 있음

```
s = d / t;
print s;
```

0이면 안됨
예외처리필요

실수(mistake)	시간이 0인 경우에 어떤 처리를 해야 하는지를 프로그래머가 고려하지 않았다.
결함(fault)	시간이 0이 되는 경우에(즉 t=0) 처리하는 코드가 없다.
에러(error)	시간이 0이 되는 경우에 예외가 발생한다.
오작동(failure)	예외가 전달되어 프로그램의 실행이 중단된다.

1.1 테스트 프로세스


테스트 계획
테스트 요구 사항 정의서
테스트 계획서
테스트 케이스 설계
테스트 케이스 설계 기법 명세서
테스트 케이스 설계 명세서
원시 테이터
테스트 실행 및 측정
테스트 측정 결과
→ 예상결과-실제결과비교
결과 분석 및 보고
테스트 케이스별 결과 분석서
소프트웨어 상태 보고서
테스트 결과 보고서
오류 추적 및 수정
오류 보고서
오류 수정 계획서
오류 수정 결과 보고서
오류가 수정된 대상물
오류 수정 보고서

1.2 테스트 케이스 (1/2)

- 테스트 케이스 :
  - 특정한 프로그램이 기능 및 요구사항을 준수하는지를 확인하기 위해 만들어진 실행 내용, 입력 값, 그리고 예상된 결과의 SET

테스트케이스 명세

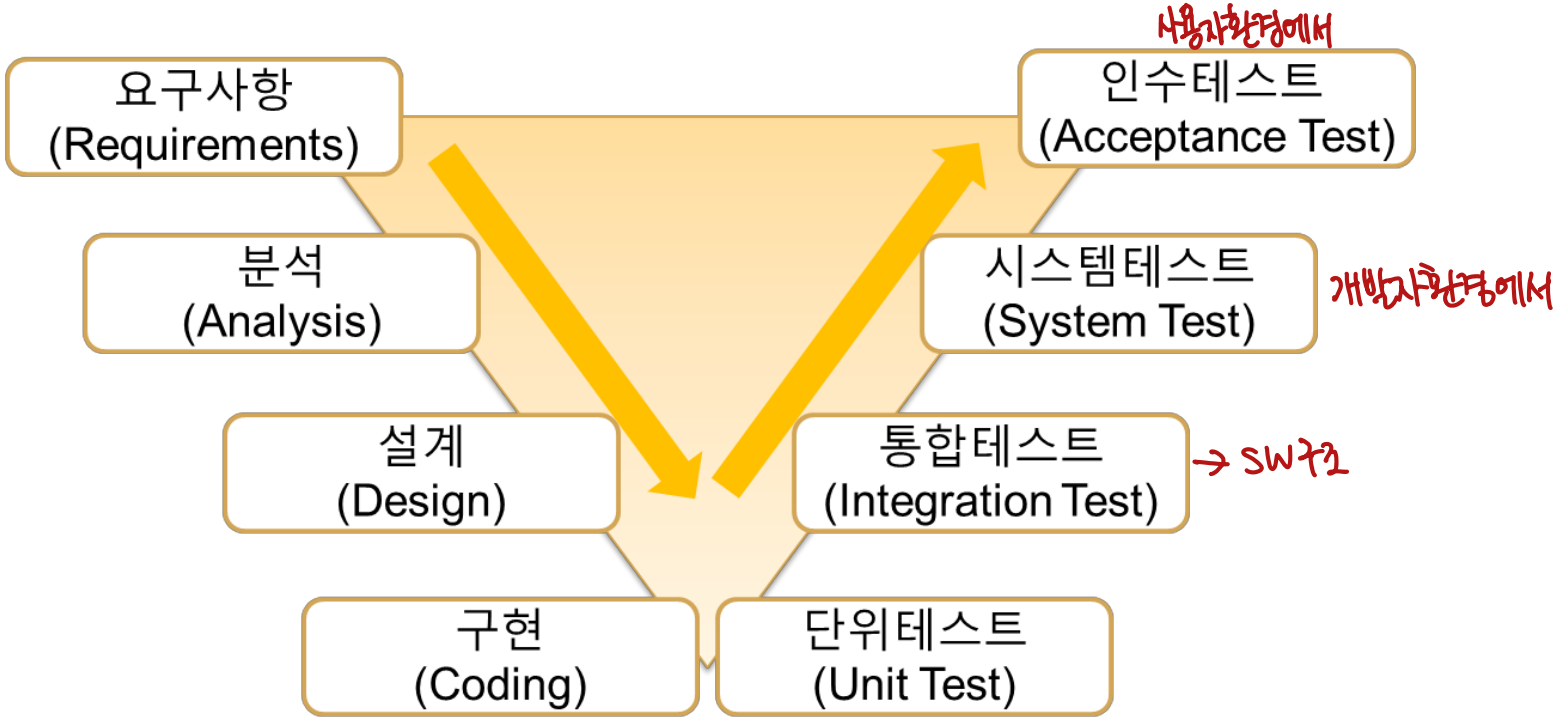
필드명	필드 설명
화면명 (프로그램명)	• 화면명에 해당하는 화면ID를 적는다. • 배치 프로그램에 해당하는 프로그램ID를 적는다.
요구 사항 ID	• 분석/설계 단계에서 생성된 요구 사항 ID 를 표기한다.
요구 사항 내용	• 요구 사항 내용을 간략히 기술한다.
관련 프로그램	• 요구 사항과 관련된 프로그램을 기술한다.
테스트 케이스명	• 테스트 케이스명은 해당 테스트 케이스를 수행하여 달성하고자 하는 목표와 관련이 있다. • 목표를 간단히 줄여서 목표로 정한다.
테스트 내용	• 테스트 내용을 적어야 하는 이유는 테스트 케이스는 축적되어, 향후 누구라도 실행이 가능해야 하기 때문이다. 따라서, 제3자가 이해할 수 있도록 객관적으로 기술되어야 한다. • 테스트 케이스에 대한 구체적인 설명을 기술하도록 한다.
테스트 데이터	• 제3자가 수행하기 위해서는 테스트 데이터도 구체적인 값이 표기되어야 한다.
예상 결과	• 확인 내용: 테스트 케이스를 실행시켰을 경우, 성공을 확인할 수 있는 값이나 메시지에 대해 기술한다.
환경 설정	• 테스트를 수행할 때 필요한 하드웨어나 소프트웨어 환경
특수 절차 요구	• 테스트 케이스 수행 시 특별히 요구되는 절차

1.2 테스트 케이스 (2/2)

- 테스트 오라클 (Test oracle)
  - 테스트 실행 결과를 검증하기 위한 메커니즘
  - 명세서로부터 추출한 옳다고 믿을 수 있는 값으로, 예상되는 결과나 그것을 구할 수 있는 수단
- 테스트 오라클 생성을 위한 소스
  - 기존 유사 프로그램의 실행 결과 활용
  - 회귀 테스트에서 사용된 테스트 결과 활용(수정에 대한 테스트인 경우)
  - 정형 명세, 즉 정형화된 수학 공식을 통한 산출 등

1.3 테스트의 단계 (1/4)

- 생명주기 모형에 따라 개발 과정의 어느 시점에서 테스트가 수행되는지에 따름



1.3 테스트의 단계 (2/4)

- 단위 테스트
- 통합 테스트
  - 단위 테스트가 종료된 모듈들을 통합하면서 수행하는 테스트
  - 모듈 간의 인터페이스 정확성이 주요 관심 사항
- 시스템 테스트
  - 개발자 환경의 통합소프트웨어 테스트 활동 → 실제로 소프트웨어가 운영될 하드웨어 환경을 갖추어 테스트 하는 시스템 테스트 활동 진행

1.3 테스트의 단계 (3/4)

- 인수 테스트
  - 인수 또는 수락 테스트
  - 사용자 환경에서 사용자가 요구한 기능을 하나씩 실행시키는 데모 형식으로 진행
- 회귀 테스트
  - 소프트웨어 테스트 과정에서 발견된 결함을 수정하고 난 후 수행하는 테스트
  - 소프트웨어 운영 과정에서 결함을 수정하거나 기능 개선으로 코드가 변경된 경우 수행

1.3 테스트의 단계 (4/4)

테스트 하니스(Test Harness)

- 테스트를 위해 생성된 코드와 데이터
- 테스트 드라이버 (test driver)와 테스트 스텁(test stub)이라고 부름
- 테스트 하니스는 완전히 테스트가 끝나면 제거

1.4 테스팅 유형

- 정적(static) 테스팅 → 실행 X
  - 소스코드 구조 확인 또는 구문/데이터 흐름을 분석
  - 워크스루 (informal), 인스펙션(formal) 등
- 동적(dynamic) 테스팅 → 실행 O
  - 프로그램 자체를 실행하여 출력이 예상대로인지 확인하는 것
  - 단위/통합/시스템/인수/회귀 테스트 등
- The 'box' approach (테스팅기법)
  - 테스트 케이스를 설계할 때 취하는 관점을 기반
  - 블랙박스 테스팅 vs. 화이트박스 테스팅

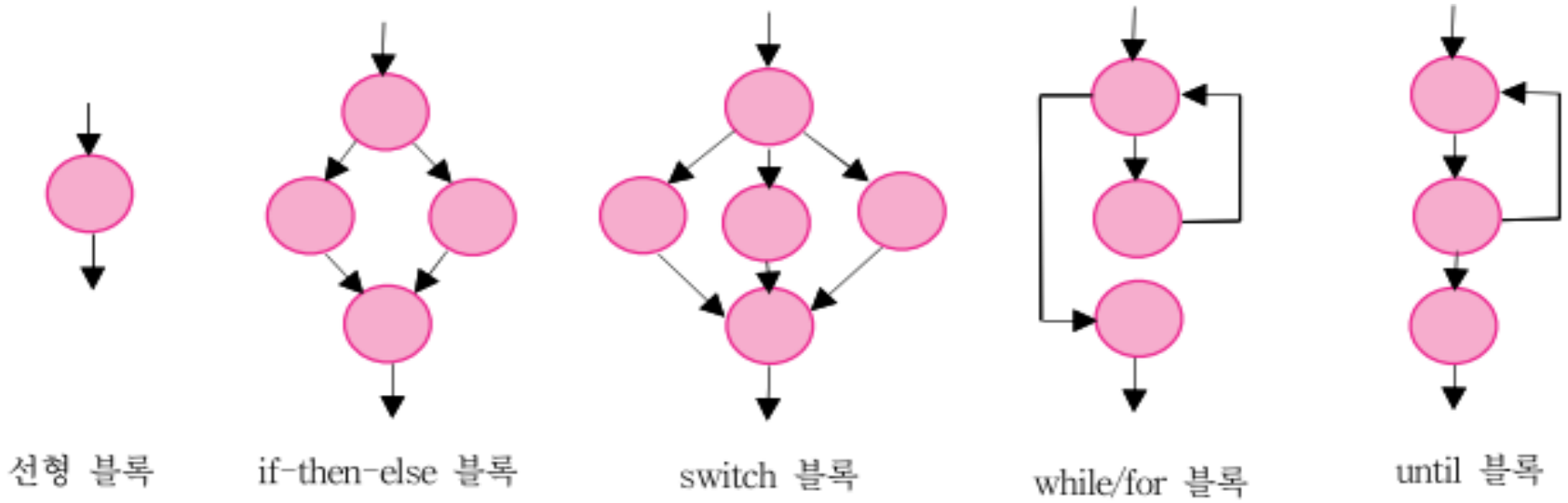
2. 코드 기반 테스트

- 화이트박스 테스팅 (Whitebox testing)
  - 모듈의 논리적인 구조를 체계적으로 점검하는 구조적 테스팅
  - 코드를 실제로 수행하지 않고, 프로그램 코드의 로직 정보를 이용하여 테스트 케이스 설계
  - 구조기반 테스트 (코드기반테스트)
  - 제어흐름 그래프(control-flow diagram)를 이용하여 주어진 검증기준을 만족하는 테스트 케이스 생성

또는 흐름그래프

2.1 제어 흐름 그래프 (1/2)

- 소스 코드를 입력하여 코드의 실행 시작부터 종료 지점 까지의 제어 흐름을 다이어그램으로 표현 한 것
- 모듈 내의 모든 세그먼트를 노드(node)로, 모듈 내의 제어흐름이 간선(edge)으로 표현

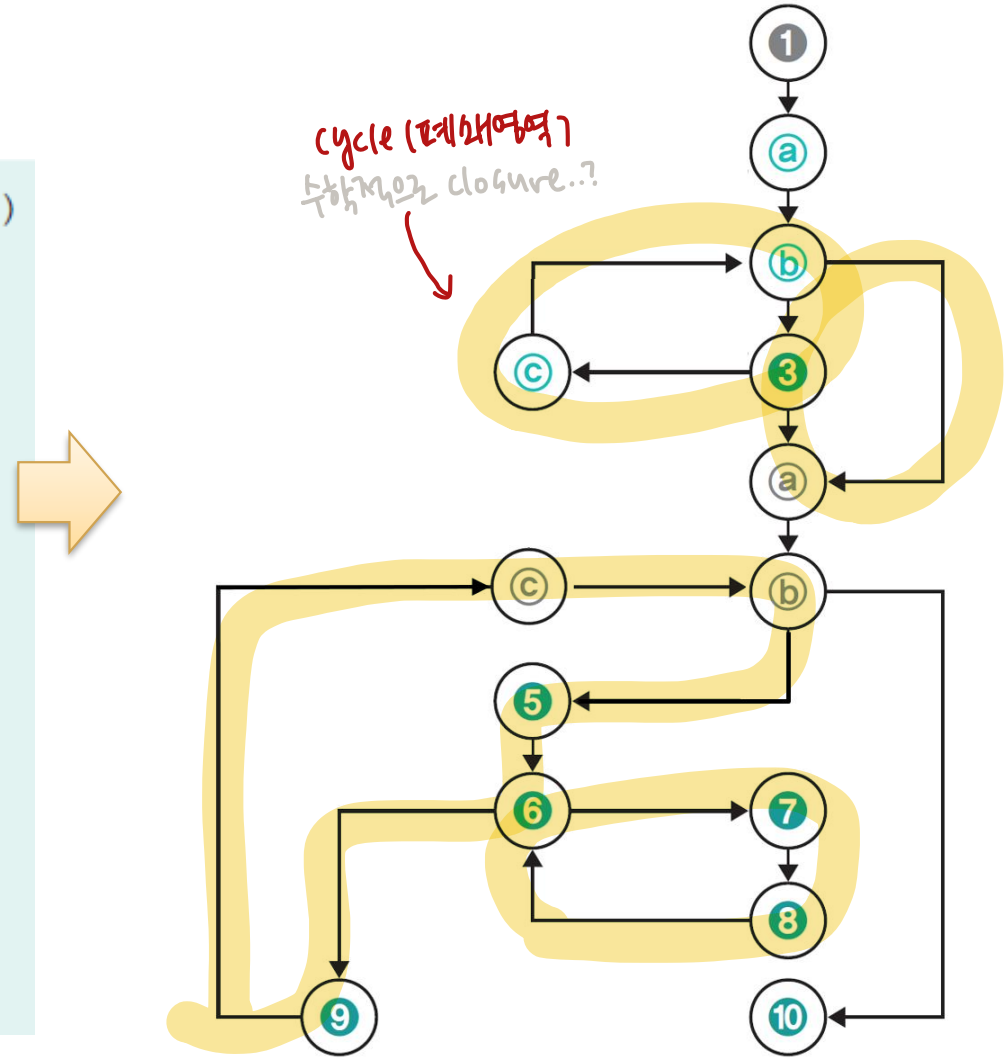


2.1 제어 흐름 그래프 (2/2)

- 예제

```
insertion_procedure (int a[], int p[], int N)
{
  ❶ int i, j, k;
  ❷ for (ⓐi=0; ⓑi<=N; ⓒi++)
  ❸     p[i] = i;
  ❹ for (ⓐi=2; ⓑi<=N; ⓒi++)

  ❺     k = p[i]; j = 1;
  ❻     while (a[p[j-1]] > a[k]) {
  ❼         p[j] = p[j-1];
  ❽         j--;
        }
  ❾     p[j] = k;
  ❿ }
}
```



+) 11/22 싸이클로매틱 복잡도 시험

4+1개 경로

2.2 검증기준 (Coverage)

- 테스트에 의해 확인되는 시스템의 범위를 말함
  - 테스트를 위한 준비 데이터가 입력으로 주어졌을 때 코드를 구성하는 문장과 제어 흐름을 모두 거쳐 가는가를 확인
  - 코드의 구조를 이루는 것은 크게 문장(Statement), 조건(Condition), 분기(Decision) ⇒ 이러한 구조를 얼마나 커버했느냐에 따라 코드 커버리지의 측정기준이 나뉨
  - 경로기반 커버리지라고도 함

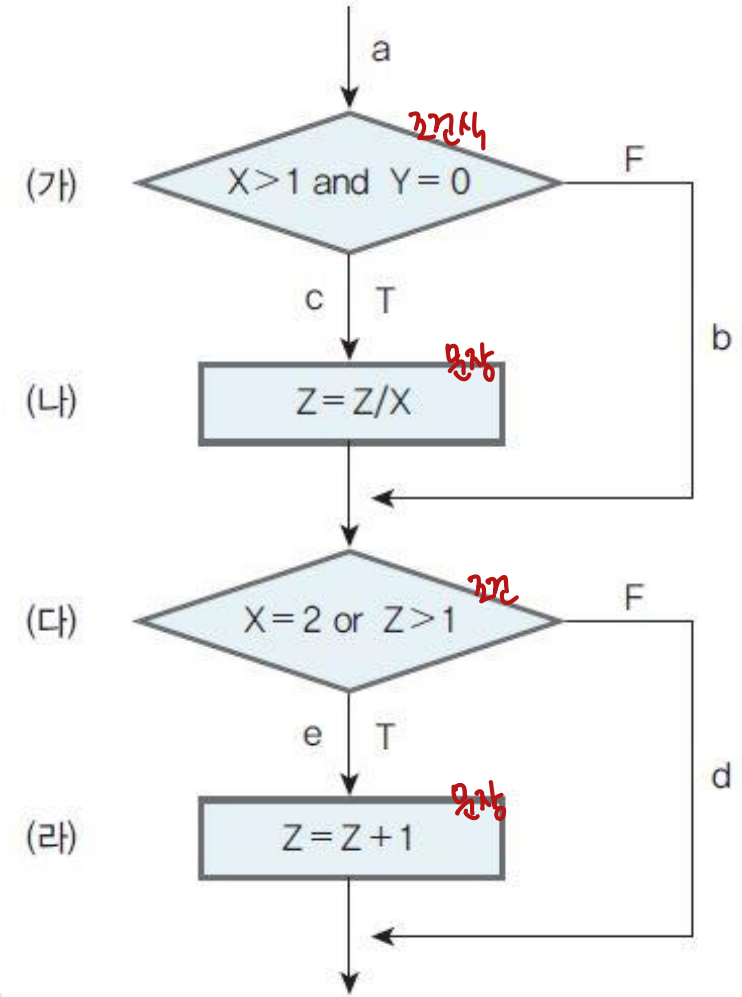
2.3 문장 커버리지 (1/4)

- 프로그램에 있는 모든 문장이 적어도 한번씩은 수행될 수 있는 경로를 선택해서 테스트 케이스 설계
  - 코드를 구성하는 모든 문장을 한 번씩 거쳐 가면 문장 커버리지를 100% 달성

- 커버리지를 100% 달성했다고 해도, 프로그램에 오류가 없음을 보장하는 것은 아님

2.3 문장 커버리지 (2/4)

- 원시코드로 제어흐름 그래프를 작성



→ 두 조건식이 모두 T여야 모든 문장이 실행됨

2.3 문장 커버리지 (3/4)

- 모든 가능한 경로를 구함

가, 다 조건만족여부	가능경로	만족여부	설명
경로1 (T, T)	a – c – e	만족	(가)(나)(다)(라) 문장을 모두 지나감
경로2 (T, F)	a – c – d	불만족	(라) 문장을 안 지나감
경로3 (F, T)	a – b – e	불만족	(나) 문장을 안 지나감
경로4 (F, F)	a – b – d	불만족	(나) (라) 문장을 안 지나감

- 문장검증 기준을 만족하는 경로 선택

선택경로	경로	만족여부	설명
경로1 (T, T)	a – c – e	만족	(가)(나)(다)(라) 문장을 모두 지나감

2.3 문장 커버리지 (4/4)

- 선택한 경로에 대한 테스트 데이터를 가지고 실행

선택경로	테스트 데이터	경로	출력 값	만족여부
경로1 (T, T)	X=2, Y=0, Z=3	a – c – e	2.5	만족

- 문장 커버리지의 문제점
  - 조건식의 오류를 발견하지 못함
    - 예를 들어 Z>1을 Z>0으로 잘못 코딩해도 오류를 발견할 수 없음
    - 원래는 or 인데 and로 코딩해도 문장검증으로는 오류를 발견할 수 없음

+) 11/27 문장커버리지 예제사진
A=50, B=60 이도록 테스트데이터설정

조건문에 대한검증

- 조건문에 대해 T와 F가 적어도 한번씩 수행할 수 있도록 테스트 ⇒ 분기 커버리지

2.4 분기 커버리지 (1/4)

- 결정 커버리지(Decision Coverage)라고도 함
- 조건문에 대해 T, F가 최소한 한번은 실행되도록 경로를 선정하는 방식
  - 분기 시점 또는 합류 위치에서 조건과 관련된 오류를 발견할 가능성이 높음
  - switch 문에서는 모든 case 문과 default 문이 선정되어야 함
  - for 문이나 while 문에서는 적어도 한 번은 루프의 내부가 실행되어야 함

2.4 분기 커버리지 (2/4)

- 제어흐름그래프에서 모든 가능한 경로를 구함

가, 다 조건만족여부	가능경로	만족여부	설명
경로1 (T, T)	a – c – e	불만족	(F, F) 경로를 테스트 안함
경로2 (T, F)	a – c – d	불만족	(F, T) 경로를 테스트 안함
경로3 (F, T)	a – b – e	불만족	(T, F) 경로를 테스트 안함
경로4 (F, F)	a – b – d	불만족	(T, T) 경로를 테스트 안함

- 모든 경로 중 분기검증기준을 만족하는 경로 선택
  - 가능한 경로들이 분기 검증기준을 만족시키지 못할 경우, 경로를 묶어서 기준을 만족시킬 수 있는 경우를 찾아야 함

2.4 분기 커버리지 (3/4)

- 경로를 묶어서 :

(1, 2)(1, 3)(1, 4)(2, 3)(2, 4)(3, 4) ⇒ (1, 4) 또는 (2, 3)

가, 다 조건만족여부	가능경로
경로1 (T, T)	a – c – e
경로4 (F, F)	a – b – d

가, 다 조건만족여부	가능경로
경로2 (T, F)	a – c – d
경로3 (F, T)	a – b – e

- 선택한 경로에 대한 테스트 데이터를 실행

선택경로	테스트 데이터	경로	출력 값	만족여부
경로1 (T, T)	X=2, Y=0, Z=10	a – c – e	6	만족
경로4 (F, F)	X=0, Y=0, Z=1	a – b – d	1	만족

2.4 분기 커버리지 (4/4)

- 경로1에서, Z>1을 Z>0으로 코딩해도 오류를 발견 못함
  - 개별 조건식이 or로 연결되므로 둘 중 하나만 만족하면 결과에 영향을 주지 않으므로

조건문 커버리지에서 확장

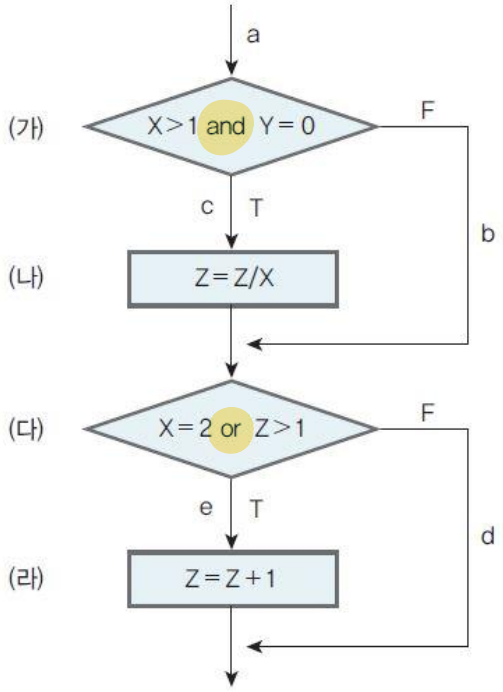
- 조건문 내의 개별 조건식에 대해 각각 T와 F인 경우를 최소한 한번씩 수행 ⇒ 조건 커버리지

+) 11/27 문장 커버리지 예제로 분기 커버리지 풀기!

선택경로	테스트데이터
(T, T)	A=50, B=60
(F, F)	A=30, B=30

2.5 조건 커버리지 (1/2)

- 조건문에 대하여 모든 가능한 결과가 적어도 한 번씩은 나타날 수 있도록 데이터 케이스를 설계
  - 진리표에 나타날 수 있는 모든 경우에 대하여 테스트 케이스를 준비



(가) : (X>1) = T and (Y=0) = T
(가) : (X>1) = T and (Y=0) = F
(가) : (X>1) = F and (Y=0) = T
(가) : (X>1) = F and (Y=0) = F

(다) : (X=2) = T or (Z>1) = T
(다) : (X=2) = T or (Z>1) = F
(다) : (X=2) = F or (Z>1) = T
(다) : (X=2) = F or (Z>1) = F

2.5 조건 커버리지 (2/2)

인접한 것끼리 묶음
원래는 가능한 모든 경우에 대해 조건식이 모두 한번씩 T, F 조합을 갖도록 수행해야 함

번호	경로	(가)	(다)	개별 조건식	두 경로의 합	테스트 케이스	전체 조건식	
1	경로 1	T, T	T, T	불만족	만족	적합	(가)T, (다)T	만족
	경로 2	F, F	F, F	불만족			(가)F, (다)F	
2	경로 3	T, T	T, F	불만족	만족	적합	(가)T, (다)T	불만족/ (다)F가 없음
	경로 4	F, F	F, T	불만족			(가)F, (다)T	
3	경로 5	T, T	F, T	불만족	만족	제외(경로 3, 4와 중복)	(가)T, (다)T	X
	경로 6	F, F	T, F	불만족			(가)F, (다)T	
4	경로 7	T, T	F, F	불만족	만족	적합	(가)T, (다)F	만족
	경로 8	F, F	T, T	불만족			(가)F, (다)T	
5	경로 9	T, F	T, T	불만족	만족	적합	(가)F, (다)T	불만족/ (가)T가 없음
	경로 10	F, T	F, F	불만족			(가)F, (다)F	
6	경로 11	T, F	T, F	불만족	만족	적합	(가)F, (다)T	불만족/(가)T, (다)F가 없음
	경로 12	F, T	F, T	불만족			(가)F, (다)T	
7	경로 13	T, F	F, T	불만족	만족	제외(경로 11, 12와 중복)	(가)T, (다)T	X
	경로 14	F, T	F, F	불만족			(가)T, (다)T	
8	경로 15	T, F	F, F	불만족	만족	적합	(가)T, (다)T	불만족/(가), (다)F가 없음
	경로 16	F, T	T, T	불만족			(가)T, (다)T	

조건커버리지 문제풀기!

선택경로	테스트데이터
(T,T)	A=50, B=60
(F,F)	A=30, B=30
(T,F)	A=40, B=65
(F,T)	A=55, B=40

3. 결함 기반 테스트 (1/5)

화이트박스테스트는 소스코드(경로)를 보고 테스트케이스 설정
블랙박스테스트는 (요구사항) 명세를 보고 테스트
→ 개발과정 中 나오는 산출물을 이용!

- 결함기반 테스트
  - 프로그래밍 과정에서 개발자의 실수가 잠재적으로 내재되었는지를 평가하기 위한 목적으로 수행하는 테스트

+ 테스트케이스가 올바른지 확인(검증)을 목적

돌연변이 ↘

- 뮤턴트 커버리지
  - 원본 코드에서 프로그래머가 실수하기 쉬운 부분에 결함을 주입하여 뮤턴트 코드를 생성
  - 원본 코드에 나타나는 요소(변수, 연산자 등)들을 다른 것으로 대체하여 뮤턴트를 생성

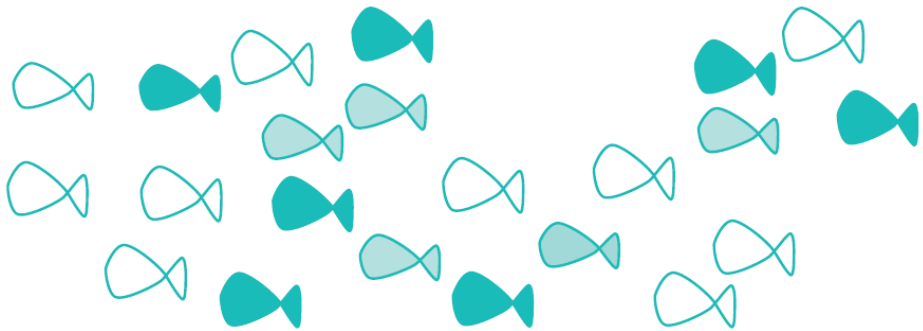
→ 원래와 다른 결과가 나와야 함!

3. 결함 기반 테스트 (2/5)

- 뮤턴트 생성자
  - 논리 연산자의 대체 (예: && 연산자를 || 연산자로 대체)
  - 관계 연산자의 대체
  - 산술 연산자의 대체
  - 문장 삭제
  - 단항 연산자의 삽입 (예: 양의 정수를 갖는 변수 X를 2X로 대체)
  - 배열 참조에 대한 대체 (예: A[i]를 A[i21]로 대체)

3. 결함 기반 테스트 (3/5)

- 뮤턴트 기반 테스트의 개념 (뮤테이션 테스팅)
  - 흰색물고기 : 정상적인 코드 요소
  - 진한 별색 물고기 : 잠재적인 결함으로 예상되는 코드 요소
  - 흐린 별색 물고기(임의로 결함이 있는 코드 요소) 주입 후 결함 물고기를 낚아내면 잠재적 결함이 함께 올라올 수 있다는 개념



3. 결함 기반 테스트 (4/5)

- 사전에 준비된 테스트케이스로 원본 프로그램과 뮤턴트 코드를 함께 실행하여 실행 결과를 비교

➢ 뮤턴트의 실행 결과가 원본 프로그램의 실행 결과와 다르면 현 테스트 케이스로 뮤턴트 프로그램을 구별해 낼 수 있다는 의미

→ 기작성된 테스트케이스들의 "테스트적정성(test adequancy)"을 평가하는기법

3. 결함 기반 테스트 (5/5)

- 뮤턴트 코드 예제