

■ UML

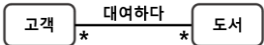
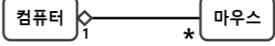
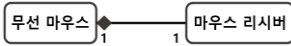

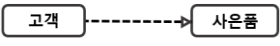

① 사물 (Things)

- 구조(Structural): 개념적, **물리적 요소**
- 행위(Behavioral): 각 요소들의 행위, **상호작용**
- 그룹(Grouping): UML 요소들을 그룹화
- 주해(Annotation): UML에 대한 부가적 설명(=주석)

② 다이어그램(Diagram)

- 구조적(Structural) 다이어그램 (=정적)
 - Class 다이어그램
 - 클래스 간의 **구조적인 관계**를 표현
 - 클래스명, 속성, 연산, 접근제어자 등으로 구성
 - Object(객체) 다이어그램
 - 클래스의 **인스턴스를 객체 간의 관계**로 표현
 - Component 다이어그램
 - 컴포넌트 간의 구성과 연결 상태** 표현
 - Deployment(배치) 다이어그램
 - 물리적 아키텍처 표현
 - ex) node, component
 - Composite Structure(복합체 구조) 다이어그램
 - 복합적인 구조를 갖는 컴포넌트, 클래스 등의 내부 구조
 - Package 다이어그램
 - 같은 그룹의 **하위 모듈을 묶어주는** 패키지 간의 의존관계
- 행위 (Behavioral) 다이어그램 (=동적)
 - Use Case 다이어그램
 - 사용자 요구사항을 분석하여 기능 중심으로 모델링한 결과물
 - Sequence 다이어그램
 - 객체들의 생성과 소멸, 객체 간 메시지 표현
 - 그 외
 - Communication(통신), State(상태), Activity(활동), Interaction Overview(상호작용), Timing

③ 관계 (Relationship)

- 연관(Association)
 - 둘 이상의 사물이 **서로 관련됨**
- 집합(Aggregation)
 - 사물이 다른 사물에 **독립적으로 포함**
- 포함(Composition)
 - 사물이 다른 사물에 **종속적으로 포함**
- 일반화(Generalization)
 - 사물이 다른 사물에 대해 **상·하위 관계**
- 의존(Dependency)
 - 필요에 의해 짧은 시간 동안만 관계 유지
- 실체화(Realization)
 - 사물의 **공통적인 기능을 상위 사물로 그룹화**

■ 모듈 응집도 / 결합도

① 응집도 (Cohesion)

- 모듈의 내부 요소들의 관계가 얼마나 밀접한지 나타냄

- 우연적 (Coincidental)**
 - 모듈 내부의 구성 요소들이 서로 아무 관련 없음
- 논리적 (Logical)**
 - 유사한 기능을 하나의 모듈에서 수행**
- 시간적 (Temporal)**
 - 각 기능들이 연관성은 없지만 **특정 시기에 함께 수행**
- 절차적 (Procedural)**
 - 하나의 문제 해결을 위해 **여러 모듈이 순차적**으로 수행
- 통신적 (Communication)**
 - 동일한 입력의 출력 결과**를 이용해 **서로 다른 기능** 수행
- 순차적 (Sequential)**
 - 모듈의 출력 결과를 **다른 모듈의 입력값**으로 사용
- 기능적 (Functional)**
 - 모든 기능의 요소들이 하나의 문제 해결을 위해 수행

★ 응집도가 높을수록(우연→기능) 높은 품질, 모듈의 독립성 ↑

② 결합도(Coupling)

- 모듈과 모듈간의 관련성이 얼마나 깊은지 나타냄

- 자료(Data)**
 - 모듈 간의 인터페이스로 전달되는 **인수와 매개변수**를 통해서만 상호작용
- 스탬프(Stamp)**
 - 관련 있는 모듈들이 **동일한 자료 구조 공유**
- 제어(Control)**
 - 전달 대상 모듈에게 **값과 제어요소**를 함께 전달
- 외부(External)**
 - 인수의 전달 없이** 다른 모듈의 내부 데이터 **참조**
- 공유(Common)**
 - 모듈 **외부**에 선언된 변수를 **참조**하여 기능 수행
- 내용(Content)**
 - 다른 모듈의 내부 기능과 데이터 **직접 사용**

★ 결합도가 낮을수록(내용→자료) 높은 품질, 모듈의 독립성 ↑

■ 테스트 커버리지

v 테스트 커버리지(Coverage)

- 주어진 테스트 케이스에 의해 수행되는 S/W의 테스트 범위를 측정하는 테스트 품질 측정 기준

① 기능 기반 커버리지

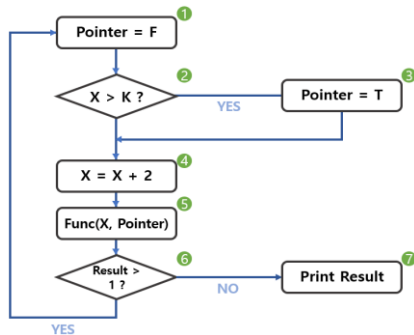
- 대상 App. 전체 기능을 모수로 설정. 실제 테스트가 수행되는 기능의 수를 측정

② 라인 커버리지

- 대상 App. 전체 소스 코드 라인 수를 모수로 설정. 테스트가 수행한 라인 수 측정

③ 코드 커버리지

- 'S/W 테스트를 충분히 진행했는가?'를 나타내는 지표



v 구문(Statement) 커버리지

- 모든 노드(구문)을 최소 한번씩 실행

<최소 수행 경로>

1 → 2 → 3 → 4 → 5 → 6 → 7

v 결정(Decision) 커버리지 = 분기(Branch) 커버리지

- 각 조건(2, 6)의 YES/NO 분기를 최소 한번씩 실행

<최소 수행 경로>

1 → 2 → 3 → 4 → 5 → 6 → 7 (2: YES | 6: NO)

1 → 2 → 4 → 5 → 6 → 1 (2: NO | 6: YES)

or

1 → 2 → 3 → 4 → 5 → 6 → 1 (2: YES | 6: YES)

1 → 2 → 4 → 5 → 6 → 7 (2: NO | 6: NO)

v 조건(Condition) 커버리지

- 각 조건식의 개별 결과값(T/F)을 최소 한번씩 실행

<최소 수행 경로>

1 → 2 → 3 → 4 → 5 → 6 → 7 (X > K : T | Result : F)

1 → 2 → 4 → 5 → 6 → 1 (X > K : F | Result : T)

or

1 → 2 → 3 → 4 → 5 → 6 → 1 (X > K : T | Result : T)

1 → 2 → 4 → 5 → 6 → 7 (X > K : F | Result : F)

v 조건/결정 커버리지

- 조건 커버리지 AND 결정(분기) 커버리지 만족

* 위의 예시에서는 분기/조건별 결과가 같으므로 결정 또는 조건 커버리지 최소 수행 경로와 같다.

v 변형 조건/결정 커버리지

- 각 조건이 최종 결과에 독립적으로 영향을 주는지 증명

<최소 수행 경로>

1 → 2 → 3 → 4 → 5 → 6 → 7 (2가 Y일 때 달라짐)

1 → 2 → 4 → 5 → 6 → 7 (2가 N일 때 달라짐)

1 → 2 → 4 → 5 → 6 → 1 ... (6이 Y일 때 달라짐)

1 → 2 → 4 → 5 → 6 → 7 (6이 N일 때 달라짐)

* 각 조건(2, 6)이 독립적으로 제어흐름에 영향을 미쳤음을 입증!

■ 관계형 DB 관련 용어

<학생>

학번	학번	이름	학과
1	20200112	김경수	메카트로닉스학과
2	20150124	김성수	지구환경과학과
3	20131216	김지은	시각디자인과

① 속성(Attribute) + 차수(Degree)

- 개체를 구성하는 고유한 특성. 테이블의 열(Column)

ex) 학년, 학번, 이름, 학과

* 속성 '학년'의 '속성값'은 1,2,3

② 도메인(Domain)

- 하나의 속성값이 가질 수 있는 모든 원자값의 집합

- 특정 속성에 대해 입력될 수 있는 값의 유형이나 범위를 의미하고 무결성을 보장하는 기준

ex) 속성 '학년'은 도메인으로 1, 2, 3을 갖음

③ 튜플(Tuple) + 기수(Cardinality)

- 하나의 개체를 표현하는 고유한 정보 단위. 테이블의 행(Row)

- '릴레이션 인스턴스' 라고도 함

ex) { 2, 20150124, '김성수', '지구환경과학과' }

④ 릴레이션(Relation)

- Attribute와 Tuple로 구성된 2차원 테이블 구조

* Relation Schema: 릴레이션에 포함된 속성명의 집합

ex) 학생(학년, 학번, 이름, 학과)

* Relation Occurrence: 릴레이션 인스턴스와 같은 의미

=릴레이션에 실제로 저장된 튜플들의 집합

⑤ 개체타입(Entity Type)

- 현실 세계의 개념을 DB상의 개념으로 표현한 것.

- 다수의 속성으로 표현

ex) '학생'의 속성: 학년, 학번, 이름, 학과

v Key

① 기본키(Primary Key): 유일성 O, 최소성 O (중복과 Null 없음)

② 후보키(Candidate Key): 유일성 O, 최소성 O, 기본키 후보

③ 대체키(Alternate Key): 기본키를 제외한 후보키들

④ 슈퍼키(Super Key): 유일성 O, 최소성 X (슈퍼키+슈퍼키 가능)

⑤ 외래키(Foreign Key): 관계된 다른 릴레이션의 기본키를 참조

* 외래키는 기본키를 참조하지만 유일성을 만족하지 않을 수 있음

■ DB 무결성 설계

① 도메인 무결성 (Domain Integrity)

- 제약대상: 속성

- 릴레이션 내 제약조건 수: 속성의 개수(=도메인)과 동일

∴ 열의 값이 정의된 도메인의 범위 안에서 표현되는 무결성

② 개체 무결성 (Entity Integrity)

- 제약대상: 튜플

- 릴레이션 내 제약조건 수: 1개

∴ 특정 열에 중복 or Null에 대한 제한을 두어 개체 식별자 역할

③ 참조 무결성 (Referential Integrity)

- 제약대상: 속성과 튜플

- 릴레이션 내 제약조건 수: 0 ~ 여러 개

∴ 참조 관계가 존재하는 두 개체간 데이터의 일관성을 보증

■ 디자인 패턴

▼ 생성 패턴(Creational)

: 클래스 정의, 객체 생성 방식에 적용 가능한 패턴

① Factory Method

- 상위 클래스: 객체 생성을 위한 인터페이스 정의
- 하위 클래스: 어떤 클래스의 인스턴스를 생성할 지 결정

② Abstract Factory

- 관련 있는 서브 클래스를 묶어서 팩토리 클래스(추상 클래스)의 조건에 따라 객체 생성

③ Singleton

- 접근제한자, 정적변수를 활용하여 클래스가 오직 하나의 인스턴스만 갖도록 하는 패턴

④ Prototype

- 동일한 타입의 객체를 생성해야 할 때 기존 객체 복사

⑤ Builder

- 복잡한 객체 생성과정을 단계별로 분리(캡슐화)하여 동일한 절차에도 서로 다른 형태의 객체 생성 가능

▼ 구조 패턴(Structural)

: 객체 간 구조와 인터페이스에 적용 가능한 패턴

① Adaptor

- 서로 다른 인터페이스를 가진 클래스 간의 변환

② Bridge

- 복잡하게 설계된 클래스를 기능부와 구현부로 분리 후 연결
- 느슨한 연결, 클래스 관계 변경 편의성
- ↔ 상속: 견고한 연결, 클래스 확장 편의성

③ Composite

- 객체들을 트리 구조로 구성, 다수의 클래스를 하나의 클래스로

④ Decorator

- 클래스 변경 없이 주어진 상황에 따라 기능 추가 가능

⑤ Facade

- 복잡한 서브 시스템들을 단순화 된 인터페이스로 제공

⑥ Proxy

- 특정 객체로의 접근을 해당 객체의 대리자를 통해 연결

▼ 행위 패턴(Behavioral)

: 기능(알고리즘), 반복적인 작업에 적용 가능한 패턴

① Iterator

- 내부 구현을 노출시키지 않고 집약된 객체에 접근하고 싶을 때

② Mediator

- 객체 간 통신이 **중재자**를 통해서 진행 → 결합도 감소

③ Interpreter

- 언어의 문법(Statement)을 해석하는 방법을 규정

④ Observer

- 객체의 상태 변화를 관찰하는 옵저버를 등록

⑤ Visitor

- 알고리즘을 자료구조에서 분리하여 클래스 수정 없이 새로운 알고리즘 추가 가능

⑥ State

- 객체 내부 상태에 따라 다른 기능 수행

■ 정규화(Normalization)

① 제 1정규화(1NF)

- **도메인**(테이블의 컬럼)이 **원자값만 가지도록** 분해

이름	취미
손흥민	게임
해리 케인	체스, 골프
베일	골프

1NF

이름	취미
손흥민	게임
해리 케인	체스
해리 케인	골프
베일	골프

② 제 2정규화(2NF)

- **부분 함수 종속 제거**(=완전 함수 종속 만족)

***완전 함수 종속**: 기본키의 부분집합이 결정자가 되지 **않음**

이름	리그	팀	골
손흥민	EPL	토트넘 홉스퍼	150
케인	분데스리가	바이에른 뮌헨	250
메시	라리가	FC 바르셀로나	400
네이마르	리그앙	파리 생제르맹	200
수아레스	라리가	FC 바르셀로나	300

2NF

이름	리그	골
손흥민	EPL	150
케인	분데스리가	250
메시	라리가	400
네이마르	리그앙	200
수아레스	라리가	300

리그	팀
EPL	토트넘 홉스퍼
분데스리가	바이에른 뮌헨
라리가	FC 바르셀로나
리그앙	파리 생제르맹
라리가	FC 바르셀로나

*기본키 "이름,리그"의 부분집합인 "리그"가 "팀"의 결정자가 되지 **않도록** 분리

③ 제 3정규화(3NF)

- **이행적 함수 종속 제거**

***이행적 종속**: A→B, B→C가 성립할 때, A→C가 성립되는 것을 의미

선수	구단	이적료
손흥민	토트넘 홉스퍼	1,100
케인	바이에른 뮌헨	1,500
메시	FC 바르셀로나	3,000
네이마르	파리 생제르맹	2,000
수아레스	FC 바르셀로나	900

3NF

선수	구단
손흥민	토트넘 홉스퍼
케인	바이에른 뮌헨
메시	FC 바르셀로나
네이마르	파리 생제르맹
수아레스	FC 바르셀로나

구단	이적료
토트넘 홉스퍼	1,100
바이에른 뮌헨	1,500
FC 바르셀로나	3,000
파리 생제르맹	2,000
FC 바르셀로나	900

④ 보이스-코드 정규화(BCNF)

- **결정자가 후보키가 아닌** 종속 제거

⑤ 제 4정규화(4NF)

- **다치 종속** 제거

⑥ 제 5정규화(5NF)

- 후보키를 통하지 않은 **조인** 종속 제거

■ 프로세스 스케줄링

- * 대기 시간: 앞선 프로세스들의 실행 시간 합 - 도착 시간
- * 평균 실행 시간: 총 실행 시간 / 프로세스 수
- * 평균 대기 시간: 총 대기시간 / 프로세스 수
- * **평균 반환 시간: 평균 실행 시간 + 평균 대기 시간**

▼ 비선점형 스케줄링

: 실행 중인 프로세스 강제 중단 불가. 일괄 처리 중심

① FIFO(First In First Out)

- 프로세스가 **도착(입력)한 순서대로 처리**. 평균 반환 시간 **대**

프로세스	실행시간	도착시간	대기시간
A	24	0	0
B	6	1	24-1 = 23
C	3	2	30-2 = 28

- * 평균 실행 시간: $(24+6+3)/3 = 11$
- * 평균 대기 시간: $(0+23+28)/3 = 17$
- * 평균 반환 시간: $11+17 = 28$

② SJF(Short Job First)

- **실행 시간이 가장 짧은 프로세스 순**으로 처리

프로세스	실행시간	도착시간	대기시간
A	24	0	0
B	6	1	27-1 = 26
C	3	2	24-2 = 22

- * 평균 실행시간: $(24+6+3)/3 = 11$
- * 평균 대기시간: $(0+26+22)/3 = 16$
- * 평균 반환시간: $11+17 = 27$

③ HRN(Highest Response-ratio Next)

- HRN 우선순위 공식 결과가 큰 작업에 높은 우선순위 부여
- *HRN 우선순위: $(\text{대기시간} + \text{실행시간}) / \text{실행시간}$

프로세스	실행시간	도착시간	대기시간	우선순위
A	20	0	0	1.0
B	4	5	20-5 = 15	4.7
C	6	10	20-10 = 10	2.6

* HRN의 대기 시간은 현재까지의 실제 대기 시간 기준

- * **각 프로세스가 끝날 때 마다 모든 큐의 우선순위를 다시 계산**
하여 가장 높은 우선순위의 프로세스를 실행

▼ 선점형 스케줄링

: 실행 중인 프로세스 강제 중단 가능. 실시간 처리 중심

① RR(Round Robin)

- **타임 퀀텀(Time Quantum)**이라는 고정 시간 단위 동안
준비 큐의 프로세스를 **순환하며 실행**하는 방식

프로세스	실행시간	도착시간	T.Q
A	5	0	2
B	3	1	
C	6	2	

<풀이과정>

시간	실행 순서	실행 결과
0 ~ 2	A 실행	A=3, B=3
2 ~ 4	B 실행	A=3, B=1, C=6
4 ~ 6	C 실행	A=3, B=1, C=4
6 ~ 8	A 실행	A=1, B=1, C=4
8 ~ 9	B 실행	A=1, B=0, C=4
9 ~ 11	C 실행	A=1, C=2
11 ~ 12	A 실행	A=0, C=2
12 ~ 14	C 실행	C=0

② SRT(Shortest Remaining Time)

- 현재 실행 중인 프로세스와 비교하여 **더 짧은 잔여 실행 시간**을
갖는 프로세스 실행

프로세스	실행시간	도착시간
A	8	0
B	4	1
C	2	2

<풀이과정>

시간	실행 순서	실행 결과
0 ~ 1	A 실행	A=7
1 ~ 2	B 실행 (A:7 > B:4)	A=7, B=3
2 ~ 4	C 실행 (B:3 > C:2)	A=7, B=3, C=0
4 ~ 7	B 실행 (A:7 > B:3)	A=7, B=0
7 ~ 14	A 실행	A=0

③ MFQ(Multilevel Feedback Queue)

- 짧게 끝나는 프로세스는 빠르게 처리하고, 오래 걸리는
프로세스는 점점 우선순위를 낮추는 방식
- 모든 프로세스는 높은 우선순위의 큐에서 시작하여 각 순회가
끝날 때 마다 다음 우선순위 큐로 강등

ex)

Queue1: 높은 우선순위 (짧은 타임 퀀텀, RR 방식 사용)

Queue2: 중간 우선순위 (보통 타임 퀀텀, RR or SJF)

Queue3: 낮은 우선순위 (긴 타임 퀀텀, FIFO)

■ 페이지 교체 알고리즘

*페이지 부재(Page Fault): 적재하려는 페이지가 페이지 프레임에 존재하지 않는 상태

① OPT(OPTimal replacement)

- 가장 오래 사용되지 "않을" 페이지 교체

*실제 운영체제에서는 미래의 참조를 알 수 없어 구현이 불가능.

이론 기준 최적의 알고리즘

② FIFO (First In First Out)

- 가장 오래된 페이지 교체

ex) 참조: [1, 3, 3, 1, 5, 4, 2, 4], 프레임: 3

<풀이과정>

페이지 프레임	1	1	1	1	1	4	4	4
		3	3	3	3	3	2	2
					5	5	5	5
페이지 부재	O	O			O	O	O	

③ SCR (Second Chance Replacement)

- FIFO 기법의 단점을 보완한 기법
- 사용한 페이지 중 자주 사용한 페이지의 교체를 방지하는 기법
- R-bit(Referenced bit, 참조 비트)가 0이되면 교체
- * R-bit는 페이지가 참조 될 때 마다 1로 설정

ex) 참조: [1, 2, 3, 2, 4, 1, 5], 프레임: 3

<풀이과정>

페이지 프레임	1 (R=1)	1	1	1	1 (R=0, 뒤로 이동) → 4 (R=1)	4 (R=1)	4 (R=1)
		2	2	2	2 (R=0, 뒤로 이동)	2 → 1 (R=1)	1 (R=1)
			3	3	3 (R=0, 뒤로 이동)	3 (R=0)	3 → 5 (R=1)
페이지 부재	O	O	O		O	O	O

④ LFU (Least Frequently Used)

- 사용 빈도가 가장 적은 페이지 교체

ex) 참조: [A, B, C, A, B, D, A, E], 프레임: 3

<풀이과정>

페이지 프레임	A (1)	A (1)	A (1)	A (2)	A (2)	A (2)	A (3)	A (2)
		B (1)	B (1)	B (1)	B (2)	B (2)	B (2)	B (2)
			C (1)	C (1)	C (1)	D (1)	D (1)	E (1)
페이지 부재	O	O	O			O		O

⑤ LRU (Least Recently Used)

- 최근에 가장 오랫동안 사용하지 않은 페이지 교체
- * 참조되지 않은 횟수가 높은 페이지 교체

ex) 참조: [A, B, B, A, C, D, E, D], 프레임: 3

<풀이과정>

페이지 프레임	A (0)	A (1)	A (2)	A (0)	A (1)	A (2)	E (0)	E (1)
		B (0)	B (0)	B (1)	B (2)	D (0)	D (1)	D (0)
					C (0)	C (1)	C (2)	C (3)
페이지 부재	O	O			O	O	O	

⑥ NUR (Not Used Recently)

- 최근에 사용하지 않은 페이지 교체

- R-bit와 M-bit(Modified bit, 수정 비트) 사용

* M-bit는 페이지에 쓰기 작업이 발생하면 1로 설정

* Clock Interrupt: 운영체제에서 특정 시간 간격(클럭 틱)마다 주기적으로 발생하는 하드웨어 인터럽트

ex) 참조: [A, B, B, A, C, D, E, D], 프레임: 3, C.I: 2

<풀이과정>

페이지 프레임	A (R=1,M=0)	A (R=1,M=0)	A (R=1,M=0)	A (R=0,M=0)	A (R=1,M=0)
		B (R=1,M=0)	B (R=1,M=0)	B (R=0,M=0)	B (R=0,M=0)
페이지 부재	O	O			
참조	A	B	B	C.I	A

A (R=1,M=0)	A (R=1,M=0)	A (R=0,M=0)	E (R=1,M=0)	E (R=1,M=0)
B (R=0,M=0)	D (R=1,M=0)	D (R=0,M=0)	D (R=0,M=0)	D (R=1,M=0)
C (R=1,M=0)	C (R=1,M=0)	C (R=0,M=0)	C (R=0,M=0)	C (R=0,M=0)
O	O		O	
C	D	C.I	E	D

* 클럭 인터럽트 직후 페이지 교체가 필요할 경우, FIFO처럼 교체 진행

+ 페이지 관리 관련 용어

- ✓ Working Set (워킹 셋)

- 프로세스가 특정 단위시간동안 자주 참조하는 페이지들의 집합

- ✓ Prepaging (프리페이징)

- 사용이 예상되는 모든 페이지를 한번에 프레임에 적재하여 초기의 과도한 페이지 부재를 방지하는 기법

- ✓ Thrashing

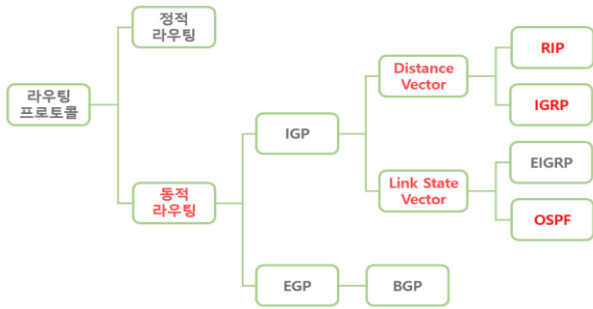
- 프로세스의 처리 시간보다 페이지 교체에 소요되는 시간이 더 많아지는 현상

- ✓ 파일 편성

- 순차(Sequential)편성, 임의(Random)편성, 색인(Indexed)편성

■ 라우팅

* 라우팅(Routing): 데이터 패킷을 전송하는데 있어 가장 빠르고 안정적인 경로를 설정하여 전송하는 기술



✓ 정적 라우팅

: 관리자가 직접 경로 설정. 빠름. 안정적. 변화에 대응 어려움

✓ 동적 라우팅

: 통신망 상태에 따라 동적 경로 설정. 라우터끼리 정보 공유

① IGP(Interior Gateway Protocol) *같은 그룹 내 정보 공유

1) Distance Vector

- 목적지까지 데이터 전송을 위한 **거리와 방향**만을 라우팅 테이블에 기록
- 인접 라우터들끼리 목적지와 매트릭(정보) 공유
- 최단 경로 탐색: Bellman-Ford 알고리즘

• RIP(Routing Information Protocol)

- **최대 15홉 이하** 규모의 네트워크 대상
- 특정 시간 간격으로 업데이트가 발생, Convergence Time이 길
- *Convergence Time: 라우터 간 변경된 정보를 주고 받는 시간

• IGRP(Internet Gateway Routing Protocol)

- CISCO사가 개발. **자율시스템(독립적 네트워크)** 내의 라우팅 데이터를 교환할 목적으로 라우터가 사용
- 홉카운트를 기준으로 정보 전송

2) Link State Vector

- 목적지까지 데이터를 전송하기 위한 **모든 경로 정보**를 라우팅 테이블에 기록
- 네트워크 변화를 빠르게 감지하여 **경로 재설정 가능**
- 최단 경로 탐색: SPF(Shortest Path First) 알고리즘

• OSPF(Open Shortest Path First)

- 모든 라우터가 **모든 링크의 정보를 파악하여 최단경로를 선출**.
- Convergence Time이 짧음

• EIGRP (Enhanced IGRP)

- IGRP 기반 개방형 라우팅 프로토콜
- 라우팅 정보 교환 및 업데이트에서 발생하는 부하를 최소화하기 위해 **부분적 업데이트**와 **제한적 업데이트 수행**
=비주기적 증분 라우팅 업데이트
- **최대 홉 카운트: 224개**

② EGP(Exterior Gateway Protocol) *다른 그룹과 정보 공유

- BGP(Border Gateway Protocol, 경계 G.P)
: **독립 운용되는 대규모 네트워크 그룹(AS) 간** 라우팅 정보 교환,
최적의 경로 선택

■ 서브네팅(Subnetting)

✓ FLSM(Fixed Length Subnet Mask)

- 동일한 크기로 서브네팅을 나누는 방식
- 각 그룹의 호스트 수가 유사한 경우 사용

✓ VLSM(Variable Length Subnet Mask)

- 동일하지 않은 크기로 서브네팅을 나누는 방식
- 여러 그룹의 호스트 수가 크게 차이나는 경우 사용

ex) IP주소: 10.20.7.15, 서브네팅 마스크: 255.255.252.0 일 때,
수신 가능한 네트워크 IP 주소는?

<풀이과정>

1. 2진수 변환

*옥텟의 비트 자리 별 10진수 값 참고! [128 | 64 | 32 | 16 | 8 | 4 | 2 | 1]

IP: 00001010.00010100.00000111.00001111 (10.20.7.15)

SM: 11111111.11111111.11111100.00000000 (255.255.252.0)

→ 10.20.7.15/22로 표현 가능 : CIDR(Classless Inter-Domain Routing)

2. IP주소와 서브네팅 마스크 AND 연산 (네트워크 주소 구하기)

00001010.00010100.00000100.00000000

= 10.20.4.0

3. 유효 호스트 주소 구하기

10.20.4.0 ~ 10.20.7.255

* 10.20.4.0 : **네트워크 주소**(사용불가)

10.20.7.255: **브로드캐스트 주소**(사용불가)

* 유효 호스트 수 구하는 공식: **2호스트 비트 수 - 2**

10.20.7.15/22의 호스트 비트 수: 32 - 22 = 10

호스트 수 = $2^{10} - 2 = 1,022$ 개

ex) IP주소: 223.13.234.132, 서브네팅 마스크: 255.255.252.192

<풀이과정>

1. 2진수 변환

132 = 128 + 4 = 10000100

192 = 128 + 64 = 11000000

→ 223.13.234.132/26로 표현 가능

→ 사용 가능한 호스트 수: $2^6 - 2 = 62$ 개

2. IP주소와 서브네팅 마스크 AND 연산 (네트워크 주소 구하기)

?.?.?.1000000 같은 형태 (SM의 C클래스까지 255이기 때문)

= 223.13.234.128 (네트워크 주소)

첫 번째 사용 가능한 호스트: 223.13.234.129

마지막 사용 가능한 호스트: 223.13.234.190 (128 + 62)

브로드캐스트 주소: 223.13.234.191 (128 + 62 + 1)

■ OSI 7 Layer

▼ 응용(Application)

- 네트워크 가상 터미널이 존재하여 서로 상이한 프로토콜에 의해 발생하는 **호환성 문제 해결**

▼ 표현(Presentation)

- 데이터 표현상의 차이 상관없이 통신 가능하도록 독립성 제공
- 응용 계층 업무 지원

▼ 세션(Session)

- **대화 제어**(통신 장치들 간의 상호작용 유지, 동기화)

▼ 전송(Transport)

- 송수신 프로세스 간의 연결
- **오류 검출 및 복구**
- **흐름 제어**
- **중복 및 누락 검사**
- 다중화

ex) TCP, UDP

▼ 네트워크(Network)

- 주소와 경로 설정하여 **패킷 전달**
- **IP 부여**를 통해 데이터를 목적지까지 빠르고 안전하게 전달

ex) 라우터

▼ 데이터링크(Data Link)

- **노드 간 송수신되는 정보의 오류와 흐름 관리**
- 정보 전달의 안정성 상승

ex) 브릿지, 스위치

▼ 물리(Physical)

- **물리적 신호**(전기, 기계)를 주고받는 계층
- 데이터의 종류나 오류를 제어하지 않음

ex) 전송 회선, 허브, 리피터

■ TCP/IP

▼ 응용 계층

- 전자우편(E-mail)
 - SMTP, POP3, MIME
- 원격제어
 - **SSH (port: 22)**: 데이터를 **암호화**하여 전달
 - **Telnet (port: 23)**: 데이터를 **평문**으로 전달
- 웹 서비스
 - **HTTP (port: 80)**: 기본형
 - S-HTTP: 제공되는 페이지만 암호화
 - **HTTPS (port: 443)**: 전체 통신 내용 암호화
- 파일전송(FTP; File Transfer Protocol)
 - Anonymous FTP, TFTP, Text Mode, Binary Mode

▼ 전송 계층

- **TCP**(Transmission Control Protocol)
 - 불안정한 IP 위에서 애플리케이션이 안정적 데이터 전송 제공
 - **3 way Handshake**
- **UDP**(User Datagram Protocol)
 - 신뢰성을 보장하지 않는 비연결성 통신 제공
 - TFTP, SNMP, RIP, NTP, RTP

▼ 인터넷 계층

- IP(Internet Protocol)
 - 패킷 교환 네트워크에서 송수신 단말기 간 정보 송수신 P.
 - **비신뢰성과 비연결성**이 특징

*패킷 전송 여부와 정확한 순서를 보장하고 싶다면 상위 프로토콜인 **TCP** 사용

- **ARP**(Address Resolution Protocol)
 - 논리적 주소(IP 주소)를 물리적 주소(MAC 주소)로 변환
 - ↔ **RARP**(Reverse ARP)
- **ICMP**(Internet Control Message Protocol)
 - IP 패킷 처리 중 발생한 **오류 메시지 수신 프로토콜**
- **DNS**(Domain Name Service)
 - 문자열로 구성된 도메인 이름을 숫자로 된 IP 주소로 변환
- **IPSec**(IP Security)
 - Network layer에서 IP패킷을 암호화하고 인증하는 등 보안을 위한 표준
 - 사설 인터넷망으로 사용할 수 있는 **VPN**을 **구현**하는데 사용
 - **IP 계층 보안성 제공**

■ 보안

▼ 악성코드

- 트로이 목마, Spyware, Worm, Key Logger, Ransomware

▼ 소극적 보안 공격

- Scanning: 네트워크상 정보를 수집하여 **취약점 파악**
- Snooping: 네트워크 상에 **떠도는 중요 정보 몰래 획득**
- Sniffing: 네트워크 상에 전송되는 **트래픽을 몰래 훑쳐보는 행위**
- Spoofing: 보안공격을 위해 **자신을 다른 주체로 속이는 행위**

▼ 서비스 거부 공격(DOS, Denial Of Service)

- SYN Flooding**: 다량의 SYN 패킷 전송하여 다른 연결 차단
- Smurf Attack**: ICMP 프로토콜 취약점 이용. **다량의 ICMP Echo Reply 전송**
- DDOS**: 분산된 다수의 좀비 PC 이용하여 공격 대상의 서비스 마비

▼ 블루투스 보안 공격

- Blue Jacking: 블루투스로 **스팸 전송**
- Blue Bugging: 블루투스로 **임의의 동작 실행**
- Blue Snarfing: 블루투스로 **임의 파일에 접근**

▼ 기타 보안 공격

- SQL Injection: SQL을 주입하여 의도치 않은 명령어 수행
- Root kit**: 추후 침입을 위한 해킹 툴 모음
- Session Hijacking**: 정당한 사용자의 세션 인증을 가로채 중요 자원에 접근
- LAND Attack**: 송수신 측 주소 동일하게 변조하여 무한 루프 발생
- XSS(Cross-Site Scripting): 웹 페이지에 **악의적 스크립트** 포함
- CSRF(Cross-Site Request Forgery): 애플리케이션의 **요청을 위조**하여 공격
- Ping of Death**: **규정된 크기 이상의 ICMP 패킷 전송**하여 Dos 유발
- Replay Attack: 중간자 공격 등으로 유출된 암호나 토큰 재사용하여 공격
- Watering Hole**: APT 공격에 주로 사용. 자주 방문하는 홈페이지의 취약점

▼ 암호 알고리즘

① 대칭키 암호화

- : 키를 사용하여 양방향으로 변환 가능 (**암호키 = 복호화키**)
- DES**(최초), **AES**(미국), IDEA(유럽), SEED(한국민간), ARIA(한국공공), RC5(블록암호화)

② 공개키 암호화

- : 키를 사용하여 양방향으로 변환 가능 (**암호키 ≠ 복호화키**)
- RSA**(최초), ECC(이산대수), DSA(미정부)

③ 해시 암호화

- : 키 없이 단방향(암호화)으로만 변환 가능
- * **Salt**(솔트): 같은 입력에 대해 **다른 출력을 얻기 위해 추가하는 무작위 문자열**
- MD(최초,데이터무결성), SHA(미국NIST)

④ 블록 암호화

- : 평문을 일정한 크기로 나누고 각 블록을 암호화

⑤ 스트림 암호화

- : 데이터 흐름을 순차적으로 암호화

■ 그 외 암기 사항

▼ 소프트웨어 개발 보안 요소(CIA)

- Confidentiality(기밀성): 인가된 사용자만 접근 가능
- Integrity(무결성): 불법적으로 생성 및 변경, 삭제되지 않음
- Availabilty(가용성): 인가된 사용자가 문제 없이 정보 사용 가능

▼ AAA

- Authentication**(검증): 신원 확인
- Authorization**(권한): 권한, 접근 범위
- Accounting**(정보수집): 사용 정보 수집

▼ 접근 통제 정책

- DAC**(임의적 접근 통제): **Identity**(신분) 기반
- MAC**(강제적 접근 통제): **Level**(보안 등급) 기반
- RBAC**(역할 기반 접근 통제): **Rule**에 접근 권한 부여. 사용자에게 역할 할당

▼ 에드혹 네트워크(Ad-Hoc Network)

- 네트워크 구성을 위한 특정 기반 구조 X (무선 인터페이스)
- 멀티 홉 라우팅 기능으로 통신거리제약극복
- 긴급 재난 시 임시 네트워크로 활용

▼ SYN

- TCP에서 두 시스템 간 정확한 메시지 전송 확인을 위해 사용하는 플래그

▼ CRC

- 데이터를 전송/저장할 때 데이터 오류 감지에 사용되는 오류 검출 코드
- 데이터에 체크섬을 추가하여 데이터를 전송하거나 저장한 후, 수신 또는 읽을 때 이 체크섬을 다시 계산하여 데이터가 변경되었는지 확인하는 기법

▼ IDS / IPS

- IDS**(Intrusion Detection System): 침입 탐지 시스템
- IPS**(Intrusion Preventing System): 침입 방지 시스템

	Firewall	IDS	IPS
목적	접근통제, 인가	침입 여부 감지	침입 이전에 방지
특징	수동적 차단 내부망 보호	로그, 시그니처 기반의 패턴매칭	정책, 규칙 DB기반의 비정상적 행위 탐지
패킷차단	O	X	O
패킷내용분석	X	O	O
오용탐지	X	O	O
오용차단	X	X	O
이상탐지	X	O	O
이상차단	X	X	O
동작 계층	전송계층 네트워크계층	응용계층 표현계층 세션계층 전송계층 네트워크계층	응용계층 표현계층 세션계층 전송계층 네트워크계층
장점	엄격한 접근통제 인가된 트래픽 허용	실시간 탐지 사후분석 대응	실시간 대응 세션 기반 탐지 가능
단점	내부자 공격 취약 네트워크 병목 현상	변형패턴은 탐지 어려움	오탐현상 발생 가능 고가 장비

정답) 45

■ C

1. 포인터 (pointer)

연산자	의미	예시
*	포인터 선언 or 역 참조 포인터가 가리키는 대상에 접근하는 연산자	*p = 10;
&	변수의 주소를 가져오는 연산자	p = &a;
[]	배열/포인터의 인덱스 접근	arr[2] = *(arr + 2)

v 포인터 예시

```
int *p;           // p는 int형을 가리키는 포인터 (스타일 차이 - C)  (포인터 선언)
int* p;           // p는 int형을 가리키는 포인터 (스타일 차이 - java & C++)

int* p1, p2;      // p1: 포인터 p2: 그냥 int
int *p1, *p2;     // p1과 p2 모두 포인터
```

```
int a = 42;
int *p = &a;      // &a : 변수 a의 주소 → p에 저장 (역참조; 포인터가 가리키는 값에 접근)
int **pp = &p;
```

```
printf("%d\n", a);    // a : 42 (실제 값)
printf("%d\n", *p);   // *p : p(= &a)가 가리키는 값 = a의 주소값이 가리키는 값 = a (= 42)
printf("%d\n", **pp); // **pp: p의 주소값(= &a)
```

```
a : 실제 값
p : &a (a의 주소값)  -> *p : a의 주소값이 가리키는 값 = a
pp : &p (p의 주소값) -> *pp : p의 주소값이 가리키는 값 = &a = p
                        -> **pp : a의 주소값이 가리키는 값 = a
```

**pp = 11;

```
printf("%d\n", a);    // 11
printf("%d\n", *p);   // 11
printf("%d\n", **pp); // 11
```

```
int arr[] = {1, 2, 3};
```

```
printf("%d\n", arr[1]); // 2
printf("%d\n", *(arr + 1)); // 2 (동일함)
```

2. -> : 포인터로 구조체 멤버에 접근할 때 쓰는 간단한 표기법

```
*예시) new_node->value = value;
      = (*new_node).value = value;
      = new_node가 가리키는 구조체의 value 멤버에 접근해서 값을 대입
```

c의 '->'는 객체의 주소(포인터)를 통해 접근 (간접 접근)
java와 python의 '.'은 객체가 실제로 존재할 때 사용 (객체 직접 접근)

3. 단일 연결 리스트: 노드가 한 방향으로만 연결될 (next)

4. 포인터 조작 prev, curr 포인터를 사용해 링크 구조를 재배치

5. 동적 메모리 malloc()으로 노드 동적 할당, free()로 메모리 반납

6. 기본 진법 표현 방식

2진수 0b1010

8진수 012 또는 0o12

10진수 10

16진수 0xA0

* 16진수 2진수 변환 (16진수 = 2진수 4자리)

16진수	2진수	10진수
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

*옥텟의 비트 자리 별 10진수 값 참고! [128 | 64 | 32 | 16 | 8 | 4 | 2 | 1]

ex) 0xA5 = 10100101 = 128+32+4+1 = 165

0xED = 11101101 = 128+64+32+8+4+1 = 237

v 2025년도 - 2회 12

```
1 #include <stdio.h>
2 #define SIZE 3
3
4 typedef struct { # Queue라는 구조체 생성 *typedef는 이름을 설정하는 옵션
5     int a[SIZE]; # Queue는 (a[SIZE], front, rear)의 구성을 갖음
6     int front;
7     int rear;
8 } Queue;
9
10 void enq(Queue* q, int val){
11     q->a[q->rear] = val; # a[rear] = val
12     q->rear = (q->rear + 1) % SIZE; # rear = (rear+1) % 3
13 }
14
15 int deq(Queue* q) {
16     int val = q->a[q->front]; # val = a[front]
17     q->front = (q->front + 1) % SIZE; # front = (front+1) % 3
18     return val;
19 }
20
21 int main() {
22     Queue q = {{0}, 0, 0};
23
24     enq(&q, 1); # a[0]=1, rear=1%3=1 → q={{1},0,1}
25     enq(&q, 2); # a[1]=2, rear=2%3=2 → q={{1,2},0,2}
26     deq(&q); # val=a[0]=1, front=1%3=1 → q={{1,2},1,2}
27     enq(&q, 3); # a[2]=3, rear=3%3=0 → q={{1,2,3},1,0}
28
29     int first = deq(&q); # val=a[1]=2, front=2 → q={{1,2,3},2,0}, first=2
30     int second = deq(&q); # val=a[2]=3, front=0 → q={{1,2,3},0,0}, second=3
31     printf("%d 그리고 %d", first, second);
32
33     return 0;
34 }
```

정답) 2 그리고 3

v 2025년도 - 2회 14

```
1 #include <stdio.h>
2
3 struct dat { int x; int y; }; # dat 구조체 생성
4
5 int main() {
6     struct dat a[] = {{1, 2}, {3, 4}, {5, 6}};
7     struct dat* ptr = a; # *ptr = a
8     struct dat** pptr = &ptr; # **pptr = &ptr → *pptr = a (ptr의 주소가 가리키는 값)
9
10    (*pptr)[1] = (*pptr)[2]; # a[1] = a[2]; → a={{1,2}, {5,6}, {3,4}}
11    printf("%d 그리고 %d", a[1].x, a[1].y); # a[1].x=5, a[1].y=6
12
13    return 0; }
```

정답) 5 그리고 6

v 2025년도 - 2회 16

풀이)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct node { int p; struct node* n; }; # 구조체 생성
5
6 int main() {
7     struct node a = {1, NULL};
8     struct node b = {2, NULL};
9     struct node c = {3, NULL};
10
11     a.n = &b; b.n = &c; c.n = NULL; # a→b→c→null 구조
12     c.n = &a; a.n = &b; b.n = NULL; # c→a→b→null 구조 (a,b의 포인터 멤버가 덮어써짐)
13     struct node* head = &c; # head=c → {3,1,{2,NULL}}
14     printf("%d %d %d", head->p, head->n->p, head->n->n->p);
15     return 0; # head[p], head[n[p]], head[n[n[p]]]
```

정답) 3 1 2

v 2025년도 - 2회 18 (오답 문제)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 struct node { # 단일 연결 리스트 노드 구조
5     char c;
6     struct node* p; # 다음 노드를 가리키는 포인터
7 };
8
9 struct node* func(char* s) {
10     struct node* h = NULL, *n; # 결과를 저장할 h 노드(NULL)와 작업이 진행될 n 노드 생성
11
12     while(*s) {
13         n = malloc(sizeof(struct node)); # 위의 선언한 n노드에 새 노드 동적 할당
14         n->c = *s++; # n[c] = s++; 현재 문자열을 저장 후, 다음 문자로 이동(후위 연산)
15             # 1회차) n={, }, s=B → n={B, }, s=E
16             # 2회차) n={, }, s=E → n={E, }, s=S
17         n->p = h; # n[p]=h: 다음 노드의 자리에 결과 노드를 가리키게 함
18             # 1회차) h=NULL n={B, NULL}
19             # 2회차) h={B, NULL }, n={E, {B, NULL }}
20
21         h = n; # 1회차) h={B, NULL}
22             # 2회차) h={E, {B, NULL }}
23     }
24     return h;
25
26 int main() {
27     struct node* n = func("BEST"); # n={T, {S, {E, {B, NULL}}}}
28
29     while(n) {
30         putchar(n->c); # n[c]를 출력
31         struct node* t = n; # 임시 노드 t 생성
32         n = n->p; # n=n[p]: 다음 노드 n으로 꺼내기
33         free(t); # 방금 사용한 노드 free로 메모리 해제 (malloc과 세트)
34     }
35     return 0;
36 }
```

정답) TSEB

v 2025년도 - 1회 10

```
1 #include <stdio.h>
2 char Data[5] = {'B', 'A', 'D', 'E'}; # Data={'B', 'A', 'D', 'E', NULL}
3 char c;
4
5 int main(){
6     int i, temp, temp2;
7
8     c = 'C';
9     printf("%d\n", Data[3]-Data[1]); # 'E'-'A'=4 출력 후 줄바꿈
10
11     for(i=0; i<5; i++){ # 0부터 4까지 1씩 증가, Data[i]가 'C'보다 클 때 까지
12         if(Data[i]>c)
13             break; # i=2 (Data[2]='D')
14     }
15
16     temp = Data[i]; # temp='D'
17     Data[i] = c; # Data={'B', 'A', 'C', 'E', NULL}
18     i++; # i=3
19
20     for(j=i<5; j++){ # i=3부터 4까지 1씩 증가, 한자리씩 뒤로 미루기
21         temp2 = Data[j];
22         Data[j] = temp;
23         temp = temp2;
24     }
25
26     for(i=0; i<5; i++){
27         printf("%c", Data[i]); # Data={'B', 'A', 'C', 'D', 'E'} 띄어쓰기 없이 출력
28     }
29 }
```

정답) 4

BACDE

v 2025년도 - 1회 11

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 void set(int** arr, int* data, int rows, int cols) {
5     for (int i = 0; i < rows * cols; ++i) {
6         arr[(i + 1) / rows] % cols = data[i]; # arr[0][1]=data[0]=5
7     }
8     # arr[0][2]=data[1]=2
9     # arr[1][0]=data[2]=7
10    # arr[1][1]=data[3]=4
11    # arr[1][2]=data[4]=1
12    # arr[2][0]=data[5]=8
13    # arr[2][1]=data[6]=3
14    # arr[2][2]=data[7]=6
15    # arr[0][0]=data[8]=9
16
17    int main() {
18        int rows = 3, cols = 3, sum = 0;
19        int data[] = {5, 2, 7, 4, 1, 8, 3, 6, 9};
20        int** arr;
21        arr = (int**) malloc(sizeof(int*) * rows);
22        for (int i = 0; i < cols; i++) {
23            arr[i] = (int*) malloc(sizeof(int) * cols); # arr 각 행에 int 배열 할당
24        }
25
26        set(arr, data, rows, cols);
27
28        for (int i = 0; i < rows * cols; i++) {
29            sum += arr[i / rows][i % cols] * (i % 2 == 0 ? 1 : -1); # i값이 홀수면 -1, 짝수면 1 곱셈.
30        } # sum = 9-5+2-7+4-1+8-3+6 = 13
31
32        for(int i=0; i<rows; i++) {
33            free(arr[i]); # 각 행 메모리 반납
34        }
35        free(arr); # 배열 메모리 반납
36
37        printf("%d", sum);
38    }
```

정답) 13

v 2025년도 - 1회 18 (오답 문제)

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 typedef struct Data {
5     int value;
6     struct Data *next;
7 } Data; # 노드를 만드는 구조체
8
9 Data* insert(Data* head, int value) { # 항상 리스트 맨 앞에 새 노드 추가
10     Data* new_node = (Data*)malloc(sizeof(Data));
11     new_node->value = value; # (*new_node).value = value; 와 같음
12     new_node->next = head; # (*new_node).next = head; 와 같음
13     return new_node;
14
15 Data* reconnect(Data* head, int value) {
16     if (head == NULL || head->value == value) return head;
17     Data *prev = NULL, *curr = head;
18     while (curr != NULL && curr->value != value) { # 특정 value값이 나올때까지 다음으로
19         prev = curr; # 1) prev=5 2) prev=4
20         curr = curr->next; # curr=(*curr).next / 1) curr=4 2) curr=3 while문 탈출
21     }
22
23     if (curr != NULL && prev != NULL) {
24         prev->next = curr->next; # (*prev).next=(*curr).next / 4→2
25         curr->next = head; # (*curr).next=head / 3→5
26         head = curr; # head=3
27     }
28
29     return head; # 위를 기반으로 노드를 재연결하면 3-5-4-2-1
30
31 int main() {
32     Data *head = NULL, *curr;
33     for (int i = 1; i <= 5; i++)
34         head = insert(head, i); # head=5 / 5-4-3-2-1=NULL
35     head = reconnect(head, 3);
36     for (curr = head; curr != NULL; curr = curr->next)
37         printf("%d", curr->value);
38     return 0; }
```

정답) 35421

v 2025년도 - 1회 19

```
1 #include <stdio.h>
2
3 typedef struct student {
4     char* name;
5     int score[3];
6 } Student;
7
8 int dec(int enc) {
9     return enc & 0xA5; # 0xA5=10100101 과 AND 연산
10 }
11
12 int sum(Student* p) {
13     return dec(p->score[0]) + dec(p->score[1]) + dec(p->score[2]); # 각 점수 AND 연산
14 }
15
16 int main() {
17     Student s[2] = { "Kim", {0xA0, 0xA5, 0xDB}, "Lee", {0xA0, 0xED, 0x81} };
18     Student* p = s; # Kim: 1010000, 10100101, 11011011
19     int result = 0; # Lee: 1010000, 11101101, 10000001
20
21     for (int i = 0; i < 2; i++) {
22         result += sum(&s[i]); # Kim 연산 결과: 160+165+129
23     } # Lee 연산 결과: 160+165+129
24     printf("%d", result); # 총합: 454 * 2 = 908
25     return 0;
26 }
```

정답) 908

v 2024년도 - 3회 7 (오답 문제)

```
1 #include <stdio.h>
2
3 int func(){
4     static int x = 0; # static 지역변수는 1. 별도의 static 영역에 저장되고
5     x+=2; # 2. 최초 실행 시점에만 초기화 진행
6     return x; # 3. 함수가 끝나도 프로그램이 끝날 때까지 값 유지
7 }
8
9 int main(){
10     int x = 1;
11     int sum=0;
12     for(int i=0;i<4;i++){ # i      0      1      2      3
13         x++; # x=2      2      3      4      5
14         sum+=func(); # static_x  2      4      6      8
15     } # sum      2      6      12     20
16     printf("%d", sum);
17
18     return 0;
19 }
```

정답) 20

v 2024년도 - 3회 12

```
1 #include <stdio.h>
2
3 struct Node {
4     int value;
5     struct Node* next;
6 };
7
8 void func(struct Node* node){
9     while(node != NULL && node->next != NULL){ # 현 node 또는 다음 node가 NULL X
10         int t = node->value; # t=1
11         node->value = node->next->value; # n1.value=3
12         node->next->value = t; # n1.next.value = n3.value=1
13         node = node->next->next; # node= node.next.next= 2 (다음 while문에서 탈출)
14     }
15 }
16
17 int main(){
18     struct Node n1 = {1, NULL};
19     struct Node n2 = {2, NULL};
20     struct Node n3 = {3, NULL};
21
22     n1.next = &n3;
23     n3.next = &n2; # 1 → 3 → 2 → NULL
24
25     func(&n1); # 3 → 1 → 2 → NULL
26
27     struct Node* current = &n1;
28
29     while(current != NULL){
30         printf("%d", current->value);
31         current = current->next;
32     }
33
34     return 0;
35
36 }
```

정답) 312

v 2024년도 - 3회 16

```
1 #include <stdio.h>
2
3 void func(int** arr, int size){
4     for(int i=0; i<size; i++){
5         *(*arr + i) = *(*arr+i) + i) % size; # *(*pp + i) = *(*pp+i) + i) % 5
6     } # *(p + i) = *(p + i) + i) % 5
7     # p[i] = (p[i] + i) % 5
8 }
9
10 int main(){
11     int arr[] = {3,1, 4, 1, 5}; # arr = &arr[0]
12     int* p = arr; # p = &arr[0]
13     int** pp = &p; # *pp = p = &arr[0] 이므로 pp = &p
14     int num = 6;
15
16     func(pp, 5); # arr = {3, 2, 1, 4, 4}
17     num = arr[2];
18     printf("%d", num);
19
20     return 0;
21 }
```

정답) 1

■ JAVA

v 2025년도 - 2회 5

```
1 public class Main {
2     public static void change(String[] data, String s){
3         data[0] = s;
4         s = "Z"; # 변경된 s는 지역 변수
5     }
6
7     public static void main(String[] args) {
8         String data[] = { "A" };
9         String s = "B";
10
11         change(data, s); # data={ "B" }
12         System.out.print(data[0] + s); # data[0]="B" , s = "B"
13     }
14 }
```

정답) BB

v 2025년도 - 2회 9 (오답 문제)

```
1 public class Main {
2
3     static interface F {
4         int apply(int x) throws Exception; # int값이 들어오면 apply() 예외처리
5     }
6
7     public static int run(F f) {
8         try {
9             return f.apply(3);
10        } catch (Exception e) {
11            return 7;
12        }
13    }
14
15    # 람다 표현식: (x) -> x+1
16    public static void main(String[] args) {
17        F f = (x) -> {
18            if (x > 2) {
19                throw new Exception();
20            }
21            return x * 2;
22        };
23
24        System.out.print(run(f) + run((int n) -> n + 9));
25    } # run(f): f의 자료형은 int가 아니므로 7 반환
26    # run(int n): int로 주었으므로 f.apply(3) >> 람다 반환값 3+9=12
```

정답) 19

v 2025년도 - 2회 10 (오답 문제)

```
1 public class Main{
2
3     public static class Parent { # x(int i)는 인스턴스 메서드로 동적 바인딩 적용
4         public int x(int i) { return i + 2; } = 실제 객체 타입(child) 기준으로 호출됨
5         public static String id() { return "P"; } # id()는 static 메서드로 정적 바인딩 적용
6         = 참조 변수의 타입(Parent) 기준으로 호출됨
7     }
8
9     public static class Child extends Parent {
10        public int x(int i) { return i + 3; }
11        public String x(String s) { return s + "R"; }
12        public static String id() { return "C"; }
13    }
14
15
16    public static void main(String[] args) {
17        Parent ref = new Child(); # 참조변수: Parent, 객체타입: Child() (오버라이딩)
18        System.out.println(ref.x(2) + ref.id()); # ref.x(2)=5, ref.id()="P"
19    }
20
21 }
```

정답) 5P

v 2025년도 - 2회 15

```
1 public class Main{
2     public static class BO {
3         public int v;
4         public BO(int v) {
5             this.v = v;
6         }
7     }
8     public static void main(String[] args) {
9         BO a = new BO(1); # a=1
10        BO b = new BO(2); # b=2
11        BO c = new BO(3); # c=3
12        BO[] arr = {a, b, c};
13        BO t = arr[0]; # t=a
14        arr[0] = arr[2]; # arr={c,b,c}
15        arr[2] = t; # t={c,b,a}
16        arr[1].v = arr[0].v; # b.v=2, c.v=3 → b.v=3
17        System.out.println(a.v + "a" + b.v + "b" + c.v); # a.v=1, b.v=3, c.v=3
18    }
19 }
```

정답) 1a3b3

v 2025년도 - 1회 5

```
1 public class Main {
2
3     public static void main(String[] args) {
4         int a=5,b=0;
5
6         try{
7             System.out.print(a/b); # a/b=5/0:
8         }catch(ArithmeticException e){ # 산술연산 유효하지 않을 때
9             System.out.print("출력1");
10        }catch(ArrayIndexOutOfBoundsException e) {
11            System.out.print("출력2");
12        }catch(NumberFormatException e) {
13            System.out.print("출력3");
14        }catch(Exception e){
15            System.out.print("출력4");
16        }finally{ # finally는 항상 실행
17            System.out.print("출력5");
18        }
19    }
```

정답) 출력1출력5

v 2025년도 - 1회 13 (오답 문제)

```
1 public class Main {
2     public static void main(String[] args) {
3         new Child(); # 1. Child 클래스 호출
4         System.out.println(Parent.total); #
5     }
6
7     class Parent { # 3. Parent 클래스 호출
8         static int total = 0; # 3-1. total=0 (static으로 고정), v=1
9         int v = 1;
10
11        public Parent() { # 4. Parent 생성자 호출
12            total += (++v); # 4-1. total=0, v=2 → 4-2. total=2, v=2
13            show(); # 5. Child show() 호출 (오버라이딩) "total += total * 2"
14        } # 5-1. total (=2) += 2 * 2 → 5-2. total = 6, v=2
15
16        public void show() {
17            total += total;
18        }
19    }
20    class Child extends Parent { # 2. Child: Parent 클래스 상속 # 6. Child 클래스 호출
21        int v = 10; # total = 6 (Parent 에서 넘어옴), v=10 (Child 클래스 변수 선언으로 초기화)
22
23        public Child() { # 7. Child 생성자 호출
24            v += 2; # 7-1. total = 6, v=12
25            total += v + +; # 7-2. total = 18, v=13
26            show(); # 8. Child show() 호출
27        } # 8-1. total = 18 + 18*2 = 54
28
29        @Override
30        public void show() {
31            total += total * 2; }
32    }
```

정답) 54

v 2025년도 - 1회 16

```
1 public class Main {
2     public static void main(String[] args) {
3         int[] data = {3, 5, 8, 12, 17};
4         System.out.println(func(data, 0, data.length - 1)); # data.length-1=4
5     }
6
7     static int func(int[] a, int st, int end) { # {3, 5, 8, 12, 17}, 0, 4
8         if (st >= end) return 0;
9         int mid = (st + end) / 2;
10        return a[mid] + Math.max(func(a, st, mid), func(a, mid + 1, end));
11    } # a[2]+MAX(f(0,2), f(3,4)) = a[2]+MAX(a[1]+a[0], a[3]) = 8+MAX(8,12) = 8+12
12 }
```

정답) 20

v 2025년도 - 1회 20

```
1 public class Main {
2     public static void main(String[] args) {
3         System.out.println(calc("5")); # 문자열 전달
4     }
5
6     static int calc(int value) { # calc(4)+calc(2) 전달 받음
7         if (value <= 1) return value;
8         return calc(value - 1) + calc(value - 2); # calc(4) = c(3)+c(2) = c(2)+c(1) + c(1)+c(0) = 3
9     } # calc(0)=0, calc(1)=1, calc(2)=1, calc(3)=2, calc(4)=3
10
11    static int calc(String str) { # 이 메서드 사용
12        int value = Integer.valueOf(str); # value=5
13        if (value <= 1) return value; # 1보다 크므로
14        return calc(value - 1) + calc(value - 3); # calc(4)+calc(2)
15    }
16 }
```

정답) 4

v 2024년도 - 3회 1

```
1 public class Main{
2     static String[] s = new String[3];
3
4     static void func(String[]s, int size){
5         for(int i=1; i<size; i++){ # 1~2
6             if(s[i-1].equals(s[i])){ # 1) s[0].equals(s[1]): 같은 문자 "A"이므로 O
7                 System.out.print("O"); # 2) s[1].equals(s[2]): 같은 문자 "A"이므로 O
8             }else{
9                 System.out.print("N");
10            }
11        }
12        for (String m : s){
13            System.out.print(m);
14        }
15    }
16
17    public static void main(String[] args){
18        s[0] = "A";
19        s[1] = "A";
20        s[2] = new String("A"); # s={"A", "A", "A"}
21
22        func(s, 3);
23    }
24 }
```

정답) OAAAA

v 2024년도 - 3회 11 (오답 문제)

```
1 public class Main{
2     public static void main(String[] args){
3         Base a = new Derivate(); # 오버라이딩된 메서드는 실제 객체 타입 기준으로 호출됨
4         Derivate b = new Derivate(); # 변수는 참조 변수의 타입기준으로 결정
5
6         System.out.print(a.getX() + a.x + b.getX() + b.x); # a.getX()=21 (동적 바인딩)
7     } # a.x=3 (정적 바인딩)
8 } # b.getX()=21 (동적 바인딩)
9 # b.x=7 (동적 바인딩)
10 class Base{
11     int x = 3;
12
13     int getX(){
14         return x * 2;
15     }
16 }
17
18 class Derivate extends Base{
19     int x = 7;
20
21     int getX(){
22         return x * 3;
23     }
24 }
```

정답) 52

v 2024년도 - 3회 18

```
1 public class ExceptionHandling {
2     public static void main(String[] args) {
3         int sum = 0;
4         try {
5             func(); # NullPointerException() 생성
6         } catch (NullPointerException e) { # 실행
7             sum = sum + 1; # sum=1
8         } catch (Exception e) {
9             sum = sum + 10;
10        } finally { # 실행
11            sum = sum + 100; # sum=101
12        }
13        System.out.print(sum);
14    }
15
16    static void func() throws Exception {
17        throw new NullPointerException();
18    }
19 }
```

정답) 101

v 2024년도 - 3회 19 (오답 문제)

```
1 class Main {
2     public static class Collection<T>{ # 제네릭 타입 소거(Type Erasure) 문제
3         T value; # 컴파일 후 T → Object
4     }
5     public Collection(T t){ value = t; } # 컴파일 후 T → Object
6
7     public void print(){ new Printer().print(value); }
8
9     class Printer{
10        void print(Integer a){ System.out.print("A" + a); }
11        void print(Object a){ System.out.print("B" + a); }
12        void print(Number a){ System.out.print("C" + a); }
13    }
14 }
15
16 public static void main(String[] args) {
17     new Collection<>().print();
18 }
19 }
```

정답) B0