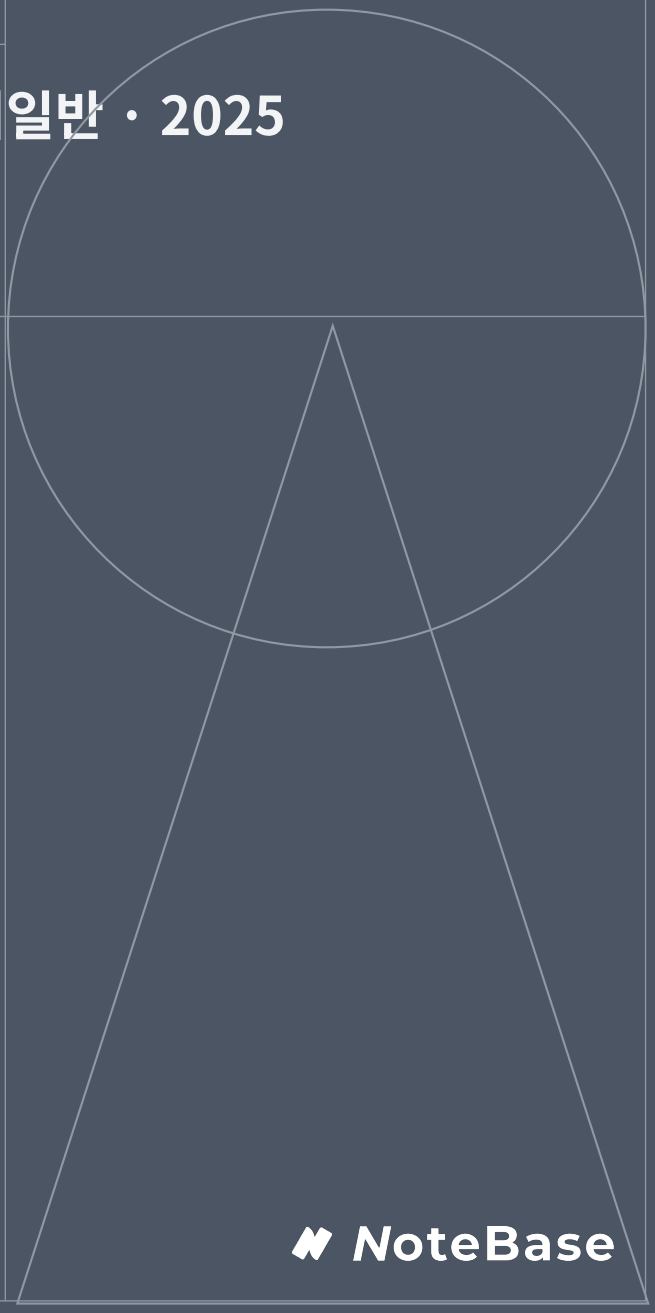
# 기출OX 해설집

공무원/지방직/9급 · 컴퓨터일반 · 2025



NoteBase

# 기출OX 해설집

## 공무원/지방직/9급 · 컴퓨터일반 · 2025

**공무원/지방직/9급 · 컴퓨터일반 · 2025**

**문 1. 데이터베이스에서 트랜잭션(transaction)의 성질이 아닌 것은?**

① 원자성(atomicity)

② 일관성(consistency)

③ 독립성(isolation)

④ 의존성(dependency)

**정답 4번**

**간단한 해설**

④의 의존성(dependency)은 트랜잭션의 기본 성질인 ACID에 포함되지 않는다. ACID의 D는 **지속성(durability)**으로, 성공적으로 완료한 트랜잭션의 결과가 장애 후에도 유지되는 성질이다.

NoteBase

**선지별 해설**

트랜잭션은 데이터베이스에서 하나의 논리적 작업 단위로 처리하는 연산들의 묶음이다. 트랜잭션이 갖추어야 할 네 가지 기본 성질을 ACID라고 한다.

① O: 원자성(atomicity)은 트랜잭션을 이루는 작업이 **모두 반영되거나 전혀 반영되지 않아야 한다**는 성질이다. 예를 들어 계좌 이체에서 출금과 입금을 하나의 트랜잭션으로 묶으면, 입금에 실패했을 때 출금만 남겨 두지 않고 출금도 취소한다.

② O: 일관성(consistency)은 트랜잭션 실행 전후에 데이터베이스가 **정해진 무결성 제약조건을 만족**해야 한다는 성질이다. 유효한 상태에서 시작한 트랜잭션은 완료 후에도 유효한 상태를 유지해야 한다.

③ O: 독립성(isolation)은 여러 트랜잭션을 동시에 실행하더라도 **각각 독립적으로 실행되는 것처럼 처리**되어야 한다는 성질이다. 다른 트랜잭션의 미완료 중간 결과가 끼어들어 실행 결과를 훼손하지 않도록 한다.
격리성이라고도 한다.

④ X: 의존성(dependency)은 트랜잭션의 네 가지 기본 성질에 속하지 않는다. ACID는 원자성·일관성·독립성·지속성을 뜻하며, D에 해당하는 성질은 **지속성(durability)**이다. 지속성은 트랜잭션이 성공적으로 완료되어 확정된 결과가 이후 장애가 발생해도 보존되어야 한다는 뜻이다.

NoteBase

**문 2. IP 주소에 대한 설명으로 옳은 것은?**

① IPv4는 8바이트 주소 체계이다.

② IPv6는 16바이트 주소 체계이다.

③ IPv6의 헤더 크기는 20바이트이다.

④ IPv4 주소가 143.248.166.12이면 클래스 C에 해당한다.

**정답 2번**

**간단한 해설**

IPv6 주소의 길이는 **128비트, 즉 16바이트**이다. 1바이트가 8비트이므로 128비트는 16바이트에 해당한다.

**선지별 해설**

① X: IPv4 주소의 길이는 32비트이므로 **4바이트**이다. 점으로 구분한 네 부분이 각각 8비트, 즉 1바이트를 나타내므로 전체가 4바이트이다.

② O: IPv6 주소의 길이는 **128비트**이다. 1바이트가 8비트이므로 16 × 8 = 128비트이며, 따라서 16바이트 주소 체계이다.

③ X: IPv6의 **기본 헤더는 40바이트로 고정**되어 있다. 필요한 부가 정보는 확장 헤더로 덧붙인다. 20바이트는 옵션이 없는 IPv4 헤더의 크기이며, IPv4 헤더는 옵션에 따라 길이가 늘어날 수 있다.

④ X: 클래스 기반 IPv4 주소 분류에서 첫 번째 옥텟이 **128~191이면 클래스 B**, 192~223이면 클래스 C이다. 143.248.166.12는 첫 번째 옥텟이 143이므로 클래스 B에 해당한다.

NoteBase

**문 3. EBCDIC(Extended Binary Coded Decimal Interchange Code)에 대한 설명으로 옳은 것만을 모두 고르면?**

ㄱ. 표현할 수 있는 문자 수는 총 256개이다.
ㄴ. 6비트를 사용하여 하나의 문자를 표시하는 방식이다.
ㄷ. 7개의 데이터 비트와 1개의 패리티 비트로 구성되어 총 8개의 비트를 사용한다.

① ㄱ
② ㄴ
③ ㄱ, ㄷ
④ ㄴ, ㄷ

**정답 1번**

**간단한 해설**

EBCDIC은 **8비트 전체로 문자 코드를 구성**하므로 256개의 코드값을 표현한다. 8비트 중 1비트를 패리티 비트로 사용하는 구성과 구별해야 한다.

**보기별 해설**

ㄱ. O: EBCDIC은 한 문자를 8비트로 나타낸다. 각 비트는 0 또는 1이므로 가능한 조합은 $2^8 = 256$개이다. 이 코드값들로 문자와 제어 기호를 표현한다.

ㄴ. X: EBCDIC은 6비트가 아니라 **8비트**를 사용하여 한 문자를 표시한다. 6비트라면 가능한 조합은 $2^6 = 64$개로, EBCDIC의 256개 코드값을 표현할 수 없다.

ㄷ. X: EBCDIC은 **8비트 전체를 문자 코드에 사용**하며, 7개의 데이터 비트와 1개의 패리티 비트로 구성되는 방식이 아니다. 7비트 ASCII에 오류 검출용 패리티 비트 1개를 붙이면 보기와 같은 8비트 구성이 된다. 이때 패리티 비트는 문자 종류를 늘리는 데 사용되지 않는다.

옳은 보기는 ㄱ뿐이므로 ㄱ만 제시한 ①에 해당한다.

NoteBase

**문 4. 다음은 입력으로 들어온 배열 [70, 11, 31, 3, 17, 9, 32, 23]에 대하여 오름차순으로 버블 정렬 알고리즘의 수행 과정을 단계별로 나타낸 것으로 (가)에 해당하는 것은?**

<초기 상태> 단계 0: [70, 11, 31, 3, 17, 9, 32, 23]
단계 1: [11, 31, 3, 17, 9, 32, 23, 70]
단계 2: [11, 3, 17, 9, 31, 23, 32, 70]
단계 3: [(가)]
□

① 3, 9, 11, 23, 17, 31, 32, 70
② 3, 11, 9, 17, 23, 31, 32, 70
③ 11, 3, 9, 17, 23, 31, 32, 70
④ 11, 3, 17, 9, 23, 31, 32, 70

**정답 2번**

**간단한 해설**

버블 정렬은 왼쪽부터 인접한 두 값을 비교하여 **앞의 값이 더 크면 교환**한다. 단계 2에서 이 과정을 한 차례 수행하면 31이 여섯 번째 자리로 이동하여 [3, 11, 9, 17, 23, 31, 32, 70]이 된다.

NoteBase

**계산 과정**

주어진 단계에서는 왼쪽부터 인접한 두 원소를 비교하여 오름차순에 어긋나면 교환한다. 한 차례 비교를 마칠 때마다 아직 자리가 확정되지 않은 값 중 가장 큰 값이 오른쪽 끝으로 이동한다. 단계 2의 배열은 [11, 3, 17, 9, 31, 23, 32, 70]이며, 오른쪽의 32와 70은 이미 자리가 확정되어 있다. 따라서 앞의 여섯 원소에 대해 다음 비교를 수행한다.

첫 번째 비교: 11과 3은 앞의 값이 더 크므로 교환한다. 배열은 [3, 11, 17, 9, 31, 23, 32, 70]이 된다.

두 번째 비교: **교환 후 두 번째 자리에 온 11**과 세 번째 자리의 17을 비교한다. 11이 더 작으므로 그대로 둔다.

세 번째 비교: 17과 9는 앞의 값이 더 크므로 교환한다. 배열은 [3, 11, 9, 17, 31, 23, 32, 70]이 된다.

네 번째 비교: 네 번째 자리의 17과 다섯 번째 자리의 31을 비교한다. 17이 더 작으므로 그대로 둔다.

다섯 번째 비교: 31과 23은 앞의 값이 더 크므로 교환한다. 배열은 [3, 11, 9, 17, 23, 31, 32, 70]이 되며, **남은 여섯 원소 중 가장 큰 31의 자리**가 확정된다.

따라서 단계 3의 (가)는 3, 11, 9, 17, 23, 31, 32, 70이며, ②의 배열과 일치한다.

NoteBase

**문 5. 프로그램 내장 방식을 적용한 컴퓨터만을 모두 고르면?**

ㄱ. EDSAC(Electronic Delay Storage Automatic Computer)
ㄴ. EDVAC(Electronic Discrete Variable Automatic Computer)
ㄷ. ENIAC(Electronic Numerical Integrator And Computer)
ㄹ. MARK I

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄷ, ㄹ

**정답 1번**

**간단한 해설**

EDSAC과 EDVAC은 **프로그램의 명령어를 내부 기억장치에 저장해 실행**하는 방식을 적용했다. 계산에 쓰는 데이터를 저장하거나 외부 종이테이프에서 명령어를 읽는 것만으로는 프로그램 내장 방식에 해당하지 않는다.

**보기별 해설**

프로그램 내장 방식은 데이터뿐 아니라 수행할 명령어도 내부 기억장치에 저장하고, 제어장치가 저장된 명령어를 읽어 실행하는 방식이다.

ㄱ. EDSAC — 프로그램 내장 방식: 수은 지연선 기억장치에 **명령어와 데이터를 저장**하여 처리했다. 프로그램을 기억장치에 넣은 뒤 그 명령어에 따라 계산을 수행한다.

ㄴ. EDVAC — 프로그램 내장 방식: 프로그램의 명령어를 데이터와 함께 내부 기억장치에 저장했다. 계산 절차를 바꿀 때 배선을 다시 연결하는 대신 **저장된 명령어를 바꾸어** 다른 작업을 수행할 수 있다.

ㄷ. ENIAC — 비내장 방식: 최초 설계에서는 명령어를 내부 기억장치에 저장하지 않고 **배선 연결과 스위치 설정**으로 연산 절차를 지정했다. 따라서 이 문항에서는 최초 설계의 ENIAC을 프로그램 내장 방식에 포함하지 않는다.

ㄹ. MARK I — 비내장 방식: 명령어를 내부 기억장치에 넣어 실행하는 대신 **천공 종이테이프에서 명령어를 차례로 읽어** 실행했다. 수치를 저장하는 장치가 있어도 프로그램 자체를 내부에 저장한 것은 아니다.

프로그램 내장 방식을 적용한 컴퓨터는 ㄱ, ㄴ이며, 이 조합은 ①에 해당한다.

NoteBase

**문 6. 운영체제에서 가변 분할 메모리 관리 기법을 사용하고 있으며, 현재 비어 있는 메모리 상태를 순서대로 표현하면 다음과 같다. 새로운 프로세스가 12 Kbyte 크기의 메모리를 요청하여 4번 위치로 할당되었다면, 이때 적용된 메모리 할당 기법은?**

| 위치 | 빈 공간 크기 |
|---|---|
| 1번 | 10 Kbyte |
| 2번 | 25 Kbyte |
| 3번 | 30 Kbyte |
| 4번 | 15 Kbyte |

① 중앙 적합(center fit)

② 최악 적합(worst fit)

③ 최적 적합(best fit)

④ 최초 적합(first fit)

**정답 3번**

**간단한 해설**

4번의 15 Kbyte는 12 Kbyte를 수용할 수 있는 빈 공간 중 **가장 작은 공간**이다. 요청을 충족하면서 남는 공간을 가장 작게 하는 방식이 **최적 적합(best fit)**이다.

NoteBase

**핵심 풀이**

할당 가능한 공간: 12 Kbyte를 요청했으므로 그 이상의 연속된 빈 공간이 필요하다. 표에서 1번은 10 Kbyte여서 부족하고, 2번 25 Kbyte·3번 30 Kbyte·4번 15 Kbyte는 할당할 수 있다.

최적 적합(best fit): **요청 크기 이상인 공간 중 가장 작은 공간**을 선택한다. 할당 가능한 세 공간 중 4번의 15 Kbyte가 가장 작고, 할당 후에는 15 − 12 = 3 Kbyte가 남는다. 따라서 4번에 할당한 조건은 최적 적합에 해당한다.

최초 적합(first fit): 주어진 순서대로 살펴보며 **처음 발견한 충분한 공간**을 선택한다. 1번은 부족하므로 건너뛰고 2번의 25 Kbyte에 할당한다.

최악 적합(worst fit): **가장 큰 빈 공간**을 선택하므로 3번의 30 Kbyte에 할당한다. 4번을 선택한 결과와는 다르다.

NoteBase

**문 7. 다음 설명에 해당하는 운영체제의 메모리 영역은?**

프로세스 내에서 함수가 호출될 때 지역변수, 매개변수와 함수가 실행을 마치고 돌아갈 주소 등을 저장하기 위한 공간이다.

① 데이터(data) 영역
② 스택(stack) 영역
③ 코드(code) 영역
④ 힙(heap) 영역

**정답 2번**

**간단한 해설**

함수 호출에 필요한 지역변수·매개변수·복귀 주소를 저장하는 곳은 **스택 영역**이다. 함수 호출과 반환에 따라 각 호출의 저장 공간이 쌓이고 해제된다.

**핵심 풀이**

일반적인 프로세스 메모리 구분에서 스택 영역은 **함수 호출마다 필요한 실행 정보**를 저장한다. 지역변수, 매개변수, 함수가 끝난 뒤 실행을 이어 갈 복귀 주소 등을 묶은 공간을 스택 프레임이라고 한다. 예를 들어 A 함수가 B 함수를 호출하면 B의 스택 프레임이 A의 프레임 위에 쌓인다. B가 끝나면 B의 프레임이 제거되고 복귀 주소를 이용해 A의 실행을 이어 간다. 이처럼 나중에 호출된 함수가 먼저 반환되는 후입선출 구조가 지문의 설명과 연결된다.

다른 영역과는 저장 대상과 용도가 다르다. 데이터 영역은 전역변수와 정적변수 등을, 코드 영역은 실행할 명령어를 저장한다.

힙 영역은 실행 중 **동적으로 할당하는 메모리**를 위한 공간으로, 함수 호출마다 복귀 주소 등의 실행 정보를 쌓아 두는 스택 영역과 구별된다.

NoteBase

**문 8. 컴퓨터 네트워크에 대한 설명으로 옳지 않은 것은?**

① TCP는 연결 지향적(connection-oriented)인 프로토콜이다.

② ARP는 IP 주소로부터 MAC 주소를 구하기 위한 프로토콜이다.

③ ICMP는 MAC 주소로부터 IP 주소를 구하기 위한 프로토콜이다.

④ IPv4에서 클래스 B는 마지막 16비트를 호스트 주소로 사용한다.

**정답 3번**

**간단한 해설**

MAC 주소로부터 IP 주소를 구하는 프로토콜은 **RARP**이다. ICMP는 주소 변환이 아니라 **IP 통신의 오류 보고와 진단**에 사용된다.

**선지별 해설**

① O: TCP는 데이터를 주고받기 전에 송신 측과 수신 측 사이에 **논리적인 연결을 설정**하는 전송 계층 프로토콜이다. 연결 상태를 유지하며 데이터의 순서와 수신 여부를 관리하므로 연결 지향적이라고 한다.

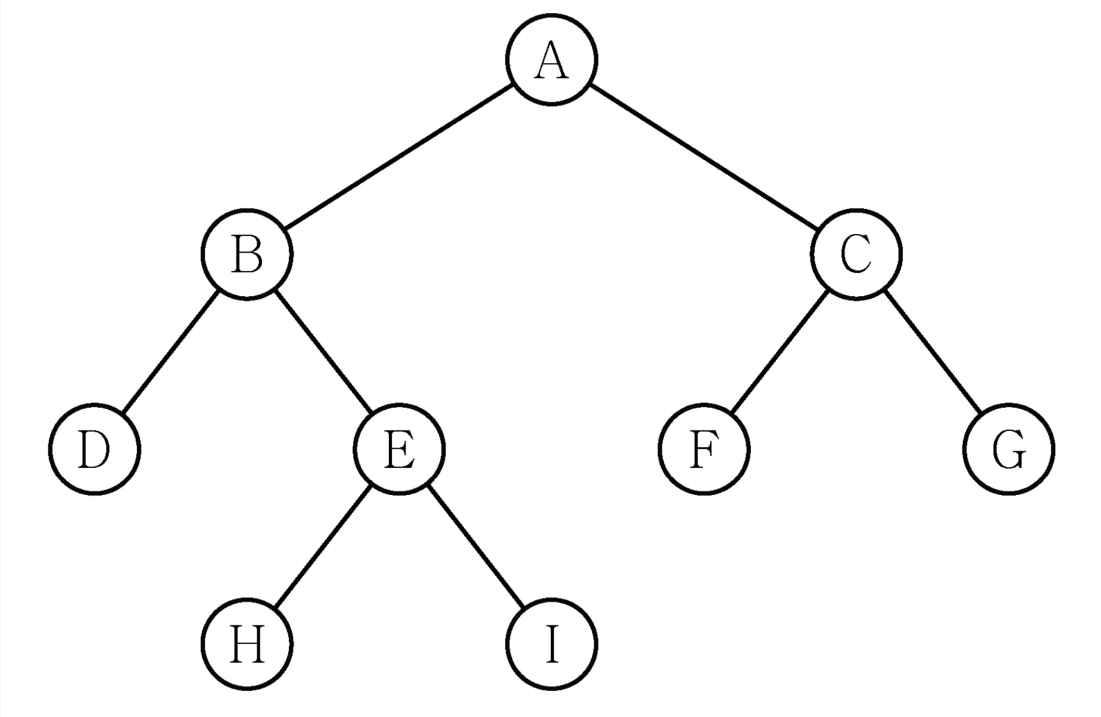
② O: ARP는 같은 링크에서 통신할 상대의 **IPv4 주소에 대응하는 MAC 주소**를 알아내는 프로토콜이다. 해당 IP 주소를 가진 장치에 MAC 주소를 묻는 요청을 보내고, 응답으로 받은 MAC 주소를 프레임 전송에 사용한다.

③ X: MAC 주소로부터 IP 주소를 구하는 기능은 **RARP**의 역할이다. ICMP는 IP 패킷 전달 중 발생한 오류를 알리거나 통신 상태를 진단하기 위한 제어 메시지를 전달한다. 예를 들어 ping은 ICMP의 에코 요청과 에코 응답을 이용해 상대 장치의 응답 여부를 확인한다.

④ O: IPv4의 **클래스 기반 주소 체계**에서 클래스 B 주소는 전체 32비트 중 앞의 16비트를 네트워크 부분으로, 뒤의 16비트를 호스트 부분으로 사용한다. 기본 서브넷 마스크는 255.255.0.0이며, 별도로 서브넷을 나누면 호스트 부분의 일부를 서브넷 구분에 사용할 수 있다.

NoteBase

**문 9. 다음 이진 트리를 중위 순회하는 경우 방문 순서는?**




① A, B, D, E, H, I, C, F, G

② A, B, D, E, H, I, F, C, G

③ D, B, H, E, I, A, F, C, G

④ D, H, I, E, B, F, G, C, A

**정답 3번**

**간단한 해설**

중위 순회에서는 각 부분 트리를 **왼쪽 부분 트리 → 현재 노드 → 오른쪽 부분 트리** 순으로 방문한다. 따라서 A는 왼쪽의 B 아래에 있는 노드까지 모두 방문한 뒤에, E는 왼쪽 자식 H와 오른쪽 자식 I 사이에 방문한다.

NoteBase

**올바른 순서**

D → B → H → E → I → A → F → C → G

D: 중위 순회는 각 부분 트리에 **왼쪽 부분 트리 → 현재 노드 → 오른쪽 부분 트리** 규칙을 적용한다. 루트 A에서 왼쪽의 B로, B에서 다시 왼쪽의 D로 내려간다. D는 자식이 없으므로 가장 먼저 방문한다.

B: 왼쪽 부분 트리인 D의 방문을 마쳤으므로 B를 방문한다. B의 오른쪽 부분 트리인 E로 이동하는 것은 그다음이다.

H: B의 오른쪽 부분 트리에서는 E에 같은 규칙을 적용한다. E를 방문하기 전에 **E의 왼쪽 자식 H**를 먼저 방문한다.

E: 왼쪽 부분 트리인 H의 방문을 마쳤으므로 E를 방문한다.

I: E를 방문한 뒤 오른쪽 자식 I를 방문한다. 이로써 B를 루트로 하는 부분 트리의 방문이 모두 끝난다.

A: **A의 왼쪽 부분 트리 전체**를 방문한 뒤에 루트 A를 방문한다. B만 방문했다고 A로 돌아오는 것이 아니라 B 아래의 E, H, I까지 방문을 마쳐야 한다.

F: A를 방문한 뒤 오른쪽 부분 트리인 C로 이동한다. C를 방문하기 전에 왼쪽 자식 F를 먼저 방문한다.

C: 왼쪽 부분 트리인 F의 방문을 마쳤으므로 C를 방문한다.

G: C를 방문한 뒤 오른쪽 자식 G를 방문한다. G는 자식이 없고 더 방문할 노드도 없으므로 순회를 마친다.

NoteBase

**문 10. 다음 설명에 해당하는 기술은?**

현실을 디지털 기반의 가상 세계로 확장해 가상 공간에서 모든 활동을 할 수 있게 만드는 기술이다. 정치, 경제, 사회, 문화 등 전반적 측면에서 현실과 비현실이 공존하는 생활형, 게임형 가상 세계라는 의미로 넓게 사용한다.

① 메타버스(metaverse)
② 사물인터넷(Internet of Things)
③ 텔레메틱스(telematics)
④ NFT(Non-Fungible Token)

**정답 1번**

**간단한 해설**

현실의 사회·경제·문화 활동을 **디지털 가상 세계로 확장**한다는 점이 메타버스의 특징이다. 단순히 기기를 연결하거나 디지털 자산을 식별하는 기술과는 구별된다.

**핵심 풀이**

메타버스는 현실과 연결된 디지털 가상 세계에서 사람들이 상호작용하며 다양한 활동을 하는 공간을 뜻한다. 예를 들어 이용자가 아바타로 모여 회의하거나 상품을 거래하는 활동을 할 수 있다. 지문의 '생활형, 게임형 가상 세계'와 '정치, 경제, 사회, 문화'는 **현실의 생활 영역이 가상 공간으로 확장**된다는 뜻이므로 메타버스에 해당한다.

사물인터넷은 **현실의 사물을 네트워크로 연결**해 데이터를 주고받는 데 초점이 있다.

텔레메틱스는 차량에 통신과 정보처리 기술을 결합해 교통·위치 정보 등의 서비스를 제공한다.

NFT는 블록체인에서 개별 디지털 자산 등을 구별하는 대체 불가능한 토큰으로, 가상 세계 안에서 활용될 수 있지만 **가상 세계 자체를 뜻하지는 않는다**.

NoteBase

**문 11. 40 KHz로 샘플링하고 10비트로 양자화하여 60초의 스테레오 음악 파일을 만들었을 때, 파일의 크기는? (단, 압축은 하지 않는다)**

① 약 2 Mbyte

② 약 6 Mbyte

③ 약 16 Mbyte

④ 약 48 Mbyte

**정답 2번**

**간단한 해설**

스테레오는 **2개 채널**을 저장하므로, 전체 데이터량은 48,000,000비트이다. 이를 바이트로 바꾸면 **약 6 Mbyte**가 된다.

**계산 과정**

샘플 수: 샘플링 주파수 40 kHz는 한 채널에서 1초에 40,000개의 샘플을 얻는다는 뜻이다. 따라서 60초 동안 한 채널의 샘플 수는 40,000 × 60 = 2,400,000개이다.

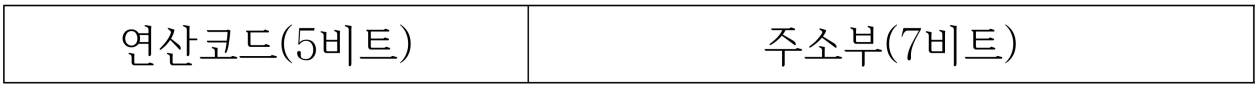
한 채널의 데이터량: 10비트 양자화는 **샘플 하나를 10비트로 표현**한다는 뜻이다. 압축하지 않으므로 한 채널에 필요한 데이터량은 2,400,000 × 10 = 24,000,000비트이다.

스테레오 데이터량: 스테레오는 **좌우 2개 채널**을 사용하므로 한 채널의 데이터량에 2를 곱한다. 전체 데이터량은 24,000,000 × 2 = 48,000,000비트이다.

바이트 환산: 1바이트는 8비트이므로 전체 비트 수를 8로 나눈다. $\frac{48,000,000}{8} = 6,000,000$ byte이므로 파일 크기는 약 6 Mbyte이며, ②의 값과 일치한다.

NoteBase

**문 12. 다음 12비트 명령어 형식에 대하여 직접 주소 지정 방식(direct addressing mode)으로 지정할 수 있는 메모리의 최대 크기는? (단, 하나의 주소공간 크기는 1바이트이다)**




① 128바이트

② 256바이트

③ 512바이트

④ 1,024바이트

**정답 1번**

**간단한 해설**

직접 주소 지정 방식에서는 **주소부의 7비트**로 메모리 위치를 지정한다. 표현할 수 있는 주소는 $2^7 = 128$개이며, 주소 하나당 1바이트이므로 최대 크기는 128바이트이다.

**계산 과정**

주소부의 길이: 그림의 12비트 명령어는 연산코드 5비트와 주소부 7비트로 나뉜다. 직접 주소 지정 방식은 **주소부의 값을 실제 접근할 메모리 주소로 사용**하므로, 주소 표현에 쓰이는 비트 수는 전체 12비트가 아니라 7비트이다.

지정 가능한 주소 수: 각 비트는 0 또는 1을 나타내므로 7비트로 표현할 수 있는 조합은 $2^7 = 128$가지이다. 주소 값은 0부터 127까지이며, 이는 서로 다른 메모리 위치 128개를 지정할 수 있다는 뜻이다.

메모리의 최대 크기: 문제에서 주소 하나에 해당하는 공간은 1바이트이다. 따라서 최대 크기는 128 × 1바이트 = 128바이트이며, ①의 128바이트에 해당한다.

NoteBase

**문 13. 값이 $(01100000)_2$가 아닌 것은? (단, 오른쪽 괄호 밖의 아래 첨자는 진법을 의미하고, +는 덧셈, ×는 곱셈 연산을 의미한다)**

① $(60)_{16}$
② $(60)_8 + (60)_8$
③ $(00110000)_2 \times (00000010)_2$
④ $(46)_{10}$에 대하여 왼쪽 산술 시프트 1회

**정답 4번**

**간단한 해설**

④의 왼쪽 산술 시프트는 오버플로가 발생하지 않으므로 46을 **2배인 92**로 만든다. 주어진 이진수 $(01100000)_2$의 십진수 값은 96이므로 서로 다르다.

**선지별 해설**

이진수의 자릿값은 오른쪽부터 1, 2, 4, 8, 16, 32, 64, 128이다. 따라서 비교 기준인 $(01100000)_2$는 64 + 32 = 96이다. 아래 O/X는 각 선지의 값이 이 기준값과 같은지를 나타낸다.

① O: 16진수 $(60)_{16}$에서 6은 **16의 자리**에 있다. 십진수로 바꾸면 6 × 16 + 0 = 96이므로 기준값과 같다.

② O: 8진수 $(60)_8$에서 6은 **8의 자리**에 있으므로 십진수로 6 × 8 = 48이다. 두 수를 더하면 48 + 48 = 96이므로 기준값과 같다.

③ O: $(00110000)_2$는 32 + 16 = 48이고, $(00000010)_2$는 2이다. 따라서 곱은 48 × 2 = 96으로 기준값과 같다.

④ X: 왼쪽 산술 시프트는 비트를 한 자리씩 왼쪽으로 옮기고 오른쪽 빈자리를 0으로 채운다. 46을 주어진 수와 같은 8비트로 나타내면 $(00101110)_2$이며, 한 번 시프트하면 $(01011100)_2$가 된다. 이 경우 오버플로가 없으므로 46 × 2 = 92이고, 이는 **96과 다른 값**이다. 따라서 이 연산의 결과는 $(01100000)_2$가 아니라 $(01011100)_2$이다.

NoteBase

**문 14. 크기가 32 Kbyte이고 4-방향 집합 연관 사상(4-way set associative mapping)을 따르는 캐시(cache)가 있다. 각 캐시 블록의 크기가 64바이트일 때, 캐시 집합의 개수는?**

① 32

② 64

③ 128

④ 256

**정답 3번**

**간단한 해설**

4-방향 집합 연관 사상은 **한 집합에 캐시 블록 4개**를 두는 방식이다. 전체 캐시 블록이 512개이므로 집합 수는 512 ÷ 4 = 128개이다.

**계산 과정**

4-방향 집합 연관 사상에서는 **한 집합이 캐시 블록 4개로 구성**된다. 따라서 전체 캐시 블록 수를 구한 뒤, 집합당 블록 수인 4로 나누면 집합 수를 얻는다.

전체 캐시 블록 수: 이 문제에서 1 Kbyte = 1,024바이트이므로 캐시 용량은 32 × 1,024 = 32,768바이트이다. 이를 블록 하나의 크기인 64바이트로 나누면 $\frac{32,768}{64} = 512$개이다. 이 512개는 **집합 수가 아니라 전체 블록 수**이다.

캐시 집합 수: 블록 4개가 한 집합을 이루므로 $\frac{512}{4} = 128$개이다. 따라서 집합의 개수는 128개이며, ③의 값과 일치한다.

NoteBase

**문 15. 다음 C 프로그램의 출력 결과는?**

```
#include <stdio.h>
int main(void) {
     int x = 100, y = 200;
     int* px = &x, *py = &y, *tmp;
     (*px) -= 10;
     (*py) += 50;
     tmp = px;
     px = py;
     py = tmp;
     (*px) += 50;
     (*py) -= 10;
     printf("x: %d, y: %d \n", x, y);
     return 0;
}
```

① x: 80, y: 300

② x: 140, y: 240

③ x: 240, y: 140

④ x: 300, y: 80

**정답 1번**

**간단한 해설**

tmp를 이용해 **포인터가 가리키는 대상**을 바꾸면 px는 y를, py는 x를 가리킨다. 이후 50은 y에 더하고 10은 x에서 빼므로 최종 값은 x = 80, y = 300이다.

NoteBase

**계산 과정**

초기 상태: x = 100, y = 200이며 px는 &x를, py는 &y를 저장한다. &는 변수의 주소를 구하고, 포인터 앞의 *는 **그 주소에 있는 변수에 접근**한다. 따라서 처음에는 *px가 x, *py가 y를 뜻한다.

첫 번째 값 변경: (*px) -= 10은 x에서 10을 빼므로 x = 100 - 10 = 90이 된다. (*py) += 50은 y에 50을 더하므로 y = 200 + 50 = 250이 된다.

포인터 교환: tmp = px에서 tmp가 x의 주소를 보관한다. px = py로 px가 y를 가리키고, py = tmp로 py가 x를 가리키게 된다. 이 세 문장은 **저장된 주소를 교환**하므로 x = 90, y = 250이라는 변수의 값은 그대로이다.

두 번째 값 변경: 이제 *px는 y를 뜻하므로 (*px) += 50에 따라 y = 250 + 50 = 300이 된다. *py는 x를 뜻하므로 (*py) -= 10에 따라 x = 90 - 10 = 80이 된다.

출력: printf는 인수 x, y의 값을 차례로 출력한다. 출력 결과는 'x: 80, y: 300'이며 ①과 일치한다.

NoteBase

**문 16. 다음과 같이 준비 상태 큐에 프로세스가 도착했을 때, 타임 슬라이스가 4 ms인 라운드 로빈(round-robin) 스케줄링으로 실행하면 프로세스의 평균 대기 시간은? (단, 프로세스 간 문맥 교환에 따른 오버헤드는 무시하고, 주어진 4개 프로세스 외에 처리할 다른 프로세스는 없다)**

| 프로세스 | 도착 시간(ms) | 실행 시간(ms) |
|---|---|---|
| $P_1$ | 0 | 8 |
| $P_2$ | 1 | 4 |
| $P_3$ | 2 | 6 |
| $P_4$ | 3 | 5 |

① 4 ms

② 11 ms

③ 18 ms

④ 44 ms

**정답 2번**

**간단한 해설**

라운드 로빈은 4 ms씩 실행한 뒤 미완료 프로세스를 **준비 큐의 맨 뒤**로 보낸다. 이 순서로 구한 대기 시간의 합은 44 ms이며, 네 프로세스의 평균 대기 시간은 **11 ms**이다.

NoteBase

**계산 과정**

라운드 로빈은 준비 큐의 앞에 있는 프로세스부터 최대 4 ms씩 실행한다. 그 안에 끝나지 않으면 **남은 실행 시간을 유지한 채 큐의 맨 뒤**로 이동한다. 문맥 교환 오버헤드는 없고, 표의 네 프로세스만 실행한다.

첫 순환: $P_1$이 0~4 ms에 실행되는 동안 $P_2$, $P_3$, $P_4$가 각각 1, 2, 3 ms에 도착한다. 따라서 4 ms의 큐 순서는 $P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_1$이다. 이어서 $P_2$는 4~8 ms에 실행되어 완료되고, $P_3$는 8~12 ms에 실행되어 2 ms가 남으며, $P_4$는 12~16 ms에 실행되어 1 ms가 남는다.

남은 실행: 16 ms의 큐 순서는 $P_1 \rightarrow P_3 \rightarrow P_4$이다. $P_1$은 16~20 ms, $P_3$는 20~22 ms, $P_4$는 22~23 ms에 남은 시간을 실행하고 완료된다. 남은 시간이 4 ms보다 짧으면 **완료 즉시 다음 프로세스가 실행**된다.

대기 시간은 준비 큐에서 CPU를 기다린 시간의 합이다. 이 문제에서는 도착부터 완료까지의 시간이 실행 시간과 대기 시간으로 나뉘므로, 대기 시간 = 완료 시각 − 도착 시각 − 실행 시간으로 계산한다. 처음 CPU를 얻기 전뿐 아니라 **실행 도중 큐로 돌아가 기다린 시간도 포함**한다.

$P_1$: 도착 시각 0 ms, 실행 시간 8 ms, 완료 시각 20 ms이므로 대기 시간은 $20 - 0 - 8 = 12$ ms이다.

$P_2$: 도착 시각 1 ms, 실행 시간 4 ms, 완료 시각 8 ms이므로 대기 시간은 $8 - 1 - 4 = 3$ ms이다.

$P_3$: 도착 시각 2 ms, 실행 시간 6 ms, 완료 시각 22 ms이므로 대기 시간은 $22 - 2 - 6 = 14$ ms이다.

$P_4$: 도착 시각 3 ms, 실행 시간 5 ms, 완료 시각 23 ms이므로 대기 시간은 $23 - 3 - 5 = 15$ ms이다.

평균 대기 시간은 네 대기 시간의 합을 프로세스 수 4로 나눈 값이다. $\frac{12 + 3 + 14 + 15}{4} = 11$ ms이므로 ②의 11 ms에 해당한다.

**문 17. 다음 파이썬 프로그램의 출력 결과는?**

```
val = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
rst = 0
for i in range(2, len(val), 3):
    rst = rst + val[i]
print(rst)
```

① 21
② 24
③ 26
④ 30

**정답 4번**

**간단한 해설**

range(2, 12, 3)에 따라 인덱스 i는 2, 5, 8, 11이 된다. 리스트의 인덱스는 **0부터 시작**하므로 해당 원소 3, 6, 9, 12를 더한 **30**이 출력된다.

**계산 과정**

반복 범위: val에는 1부터 12까지 총 12개의 원소가 있으므로 len(val)은 12이다. range(2, 12, 3)은 2부터 3씩 증가하되 **끝값 12는 포함하지 않는다**. 따라서 i는 2 → 5 → 8 → 11 순서로 바뀌고 반복문은 네 번 실행된다.

원소 선택: 파이썬 리스트의 인덱스는 **0부터 시작**한다. 따라서 val[2]는 세 번째 원소인 3이며, 같은 방식으로 val[5] = 6, val[8] = 9, val[11] = 12이다. 반복문에서 더하는 것은 인덱스 i 자체가 아니라 val[i]에 저장된 원소이다.

누적 계산: rst는 합계를 저장하는 변수이며 처음에는 0이다. rst = rst + val[i]는 기존 합계에 현재 원소를 더해 다시 저장한다. 각 반복이 끝날 때 rst는 0 + 3 = 3, 3 + 6 = 9, 9 + 9 = 18, 18 + 12 = 30으로 바뀐다.

출력: print(rst)는 반복문 밖에 있으므로 네 번의 누적 계산이 모두 끝난 뒤 최종값 30을 한 번 출력한다. 이는 ④의 30과 일치한다.

NoteBase

**문 18. 애자일(agile) 방법론에 대한 설명으로 옳지 않은 것은?**

① 고객의 요구 사항 변화를 허용하는 개발 방식이다.

② 익스트림 프로그래밍(XP)은 애자일 기법 중 하나이다.

③ 구현 단계에서는 고객의 추가 요구 사항을 반영하지 않는다.

④ 포괄적인 문서보다는 작동하는 프로그램에 더 가치를 둔다.

**정답 3번**

**간단한 해설**

애자일은 구현 중에도 **고객의 추가 요구 사항과 변경을 수용**하여 개발에 반영한다. 처음 정한 요구 사항을 고정하기보다 고객의 피드백에 따라 제품을 개선한다.

**선지별 해설**

① O: 애자일은 짧은 개발 주기를 반복하며 작동하는 소프트웨어를 점진적으로 완성하는 방식이다. 각 주기에서 고객의 피드백을 받아 **요구 사항의 변화에 대응**하므로, 처음의 요구 사항이 계속 유지되어야 한다고 전제하지 않는다.

② O: 익스트림 프로그래밍(XP)은 **애자일 개발 방법 중 하나**이다. 짧은 주기의 개발과 지속적인 피드백을 중시하며, 테스트 주도 개발과 짝 프로그래밍 등의 실천 방법을 활용한다.

③ X: 애자일은 **구현 중에도 요구 사항 변경을 수용**한다. 따라서 '구현 단계에서는 고객의 추가 요구 사항을 반영하지 않는다'는 설명은 잘못되었다. 추가 요구 사항은 고객과 협의하여 우선순위를 조정하고 이후 개발 주기에 반영할 수 있다.

④ O: 애자일은 포괄적인 문서를 갖추는 것보다 **실제로 작동하는 소프트웨어**를 더 중시한다. 작동하는 결과물을 통해 고객이 기능을 확인하고 피드백을 줄 수 있기 때문이다. 문서의 가치를 부정하거나 필요한 문서까지 작성하지 않는다는 뜻은 아니다.

NoteBase

**문 19. 객체지향 설계에서의 SOLID 원칙 중 '구체화가 아닌 추상화에 의존해야 한다'는 원칙은?**

① 개방-폐쇄 원칙(open-closed principle)

② 단일 책임 원칙(single responsibility principle)

③ 인터페이스 분리 원칙(interface segregation principle)

④ 의존관계 역전 원칙(dependency inversion principle)

**정답 4번**

**간단한 해설**

**의존관계 역전 원칙**은 구체적인 구현 클래스에 직접 의존하지 않고, 인터페이스나 추상 클래스 같은 **추상화에 의존**하도록 하는 원칙이다. 이를 통해 구현이 바뀌어도 그 기능을 사용하는 코드에 미치는 영향을 줄인다.

**개념 구분**

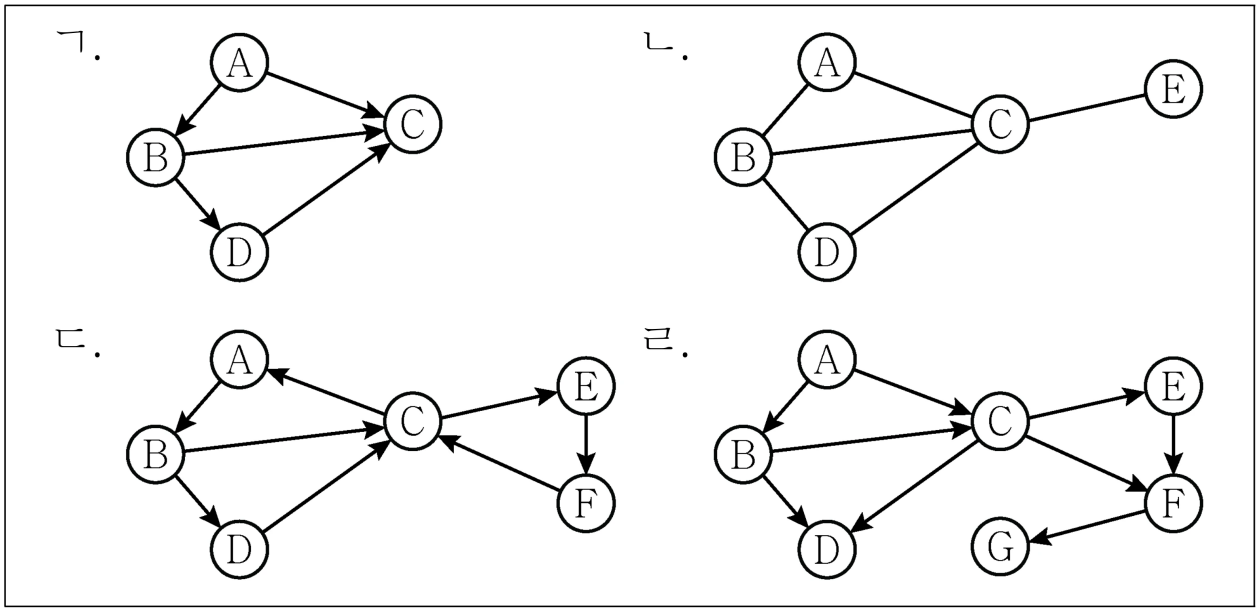
의존관계 역전 원칙: 상위 수준의 정책을 담당하는 모듈과 하위 수준의 세부 구현 모듈이 **모두 추상화에 의존**해야 한다. 또한 추상화가 세부 구현에 의존하는 것이 아니라 세부 구현이 추상화에 의존해야 한다. 따라서 문제의 '구체화가 아닌 추상화에 의존해야 한다'는 설명은 의존관계 역전 원칙에 해당한다.

예를 들어 주문 처리 코드가 특정 카드 결제 클래스에 직접 의존하는 대신 '결제' 인터페이스를 사용하도록 설계할 수 있다. 카드 결제와 계좌 이체 클래스가 이 인터페이스를 구현하면, 주문 처리 코드는 같은 인터페이스를 통해 서로 다른 결제 방식을 사용할 수 있다.

개방-폐쇄 원칙과의 구분: 개방-폐쇄 원칙은 **확장에는 열려 있고 수정에는 닫혀 있어야 한다**는 원칙이다.
추상화를 활용하면 두 원칙을 함께 실현할 수 있지만, 문제처럼 무엇에 의존해야 하는지를 직접 규정하는 것은 의존관계 역전 원칙이다.

NoteBase

**문 20. 위상 정렬(topological sort)이 가능한 그래프만을 모두 고르면?**




① ㄱ, ㄴ

② ㄱ, ㄹ

③ ㄴ, ㄷ

④ ㄷ, ㄹ

**정답 2번**

**간단한 해설**

위상 정렬은 모든 화살표의 출발 정점이 도착 정점보다 앞서도록 정점을 나열하는 것으로, **사이클이 없는 방향 그래프(DAG)**에서만 가능하다.

NoteBase

**보기별 해설**

위상 정렬은 방향 간선이 나타내는 선후 관계를 모두 지키면서 정점을 일렬로 나열하는 것이다. 예를 들어 A → B라는 간선이 있으면 A를 B보다 먼저 배치해야 한다. 방향 사이클이 있으면 서로 먼저 나와야 한다는 요구가 충돌하므로 위상 정렬이 불가능하다.

ㄱ O: 방향 간선은 A → B, A → C, B → C, B → D, D → C이며, **방향 사이클이 없다**. A → B → D → C 순으로 나열하면 모든 간선의 출발 정점이 도착 정점보다 앞에 오므로 위상 정렬이 가능하다.

ㄴ X: 간선에 화살표가 없는 **무방향 그래프**이다. 간선이 정점 사이의 선후 관계를 나타내지 않으므로 위상 정렬의 대상이 아니다. 위상 정렬이 가능하려면 방향이 주어져 있고 방향 사이클이 없어야 한다.

ㄷ X: C → A → B → C와 C → E → F → C라는 **방향 사이클**이 있다. 첫 번째 사이클만 보아도 C는 A보다, A는 B보다, B는 C보다 앞에 와야 하므로 선후 관계를 동시에 만족시킬 수 없다. 위상 정렬이 가능하려면 이러한 방향 사이클이 없어야 한다.

ㄹ O: 화살표를 따라 출발 정점으로 되돌아오는 **방향 사이클이 없다**. A → B → C → D → E → F → G 순으로 나열하면 A는 B·C보다, B는 C·D보다, C는 D·E·F보다, E는 F보다, F는 G보다 앞에 놓여 모든 방향 간선의 선후 관계를 만족한다. 따라서 위상 정렬이 가능하다.

위상 정렬이 가능한 그래프는 ㄱ과 ㄹ이므로 ②의 조합과 일치한다.

NoteBase