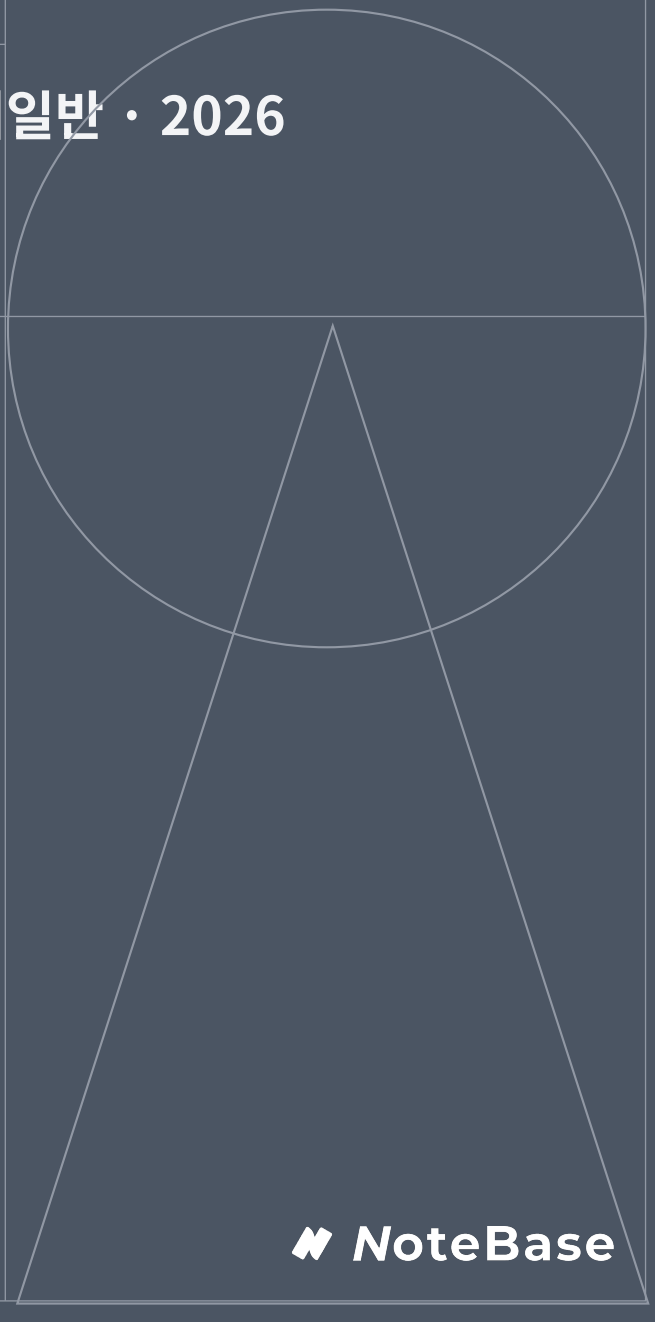
기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2026



NoteBase

기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2026

공무원/국가직/9급 · 컴퓨터일반 · 2026

문 1. 기억장치를 접근 속도가 빠른 것부터 순서대로 바르게 나열한 것은?

① 캐시 → 레지스터 → 주기억장치

② 레지스터 → 캐시 → 주기억장치

③ 레지스터 → 주기억장치 → 캐시

④ 주기억장치 → 캐시 → 레지스터

정답 2번

간단한 해설

레지스터는 CPU가 연산에 직접 사용하는 값을 보관하며, 세 기억장치 중 **접근 속도가 가장 빠르다**. 캐시는 주기억장치보다 빠른 저장 공간으로, CPU가 주기억장치에 접근하며 기다리는 시간을 줄인다.

올바른 순서

레지스터 → 캐시 → 주기억장치

레지스터: CPU 내부에서 연산에 사용할 데이터나 주소, 중간 결과 등을 보관한다. CPU가 명령을 실행할 때 직접 사용하는 작은 저장 공간으로, 캐시보다 접근 속도가 빠르다.

캐시: 주기억장치의 데이터나 명령어 중 자주 사용되거나 곧 사용될 것을 임시로 보관한다. 보통 고속의 SRAM으로 구성되어 **레지스터보다는 느리지만 주기억장치보다는 빠르다**. 필요한 내용이 캐시에 있으면 상대적으로 느린 주기억장치에 접근하는 시간을 줄일 수 있다.

주기억장치: 현재 실행 중인 프로그램과 데이터를 보관하는 공간이다. 보통 DRAM으로 구성되며, 캐시보다 큰 용량을 제공하지만 **접근 속도는 캐시보다 느리다**.

NoteBase

문 2. CPU의 연산을 처리하기 위한 데이터의 기본 단위로서 CPU가 한 번에 처리할 수 있는 데이터 크기를 나타내는 것은?

① 비트(bit)

② 바이트(byte)

③ 워드(word)

④ 니블(nibble)

정답 3번

간단한 해설

CPU가 연산할 때 기본적으로 한 번에 처리하는 데이터 단위를 **워드(word)**라고 한다.

공통 해설

워드(word)는 CPU가 연산을 위해 하나의 단위로 다루는 비트들의 묶음이다. 이 묶음에 포함되는 비트 수를 '워드 길이'라고 하며, CPU의 구조에 따라 달라질 수 있다. 문제는 특정 비트 수가 아니라 **CPU가 한 번에 처리하는 데이터 단위의 명칭**을 묻고 있으므로 워드에 해당한다.

NoteBase

문 3. 특정 환경에서 선택 가능한 행동들 중 보상(reward)을 최대화하는 행동을 선택하는 머신러닝(machine learning) 기법은?

① 지도 학습(supervised learning)

② 강화 학습(reinforcement learning)

③ 비지도 학습(unsupervised learning)

④ 준지도 학습(semi-supervised learning)

정답 2번

간단한 해설

행동의 결과로 받은 **보상**을 바탕으로 더 큰 보상을 얻는 행동을 학습하는 방식은 강화 학습이다.

공통 해설

강화 학습(reinforcement learning)은 학습 주체인 에이전트가 환경에서 행동하고, 그 결과로 받은 보상을 이용해 행동 선택 방식을 개선하는 기법이다. 보상은 행동의 결과가 얼마나 바람직한지를 알려 주는 평가 신호이며, 에이전트는 **장기적으로 얻을 누적 보상을 최대화**하도록 학습한다. 이때 어떤 상태에서 어떤 행동을 선택할지 정하는 방식을 정책이라고 한다. 문제에서 제시한 '환경에서 선택 가능한 행동 중 보상을 최대화하는 행동을 선택한다'는 설명은 이러한 강화 학습의 목표에 해당한다.

NoteBase

문 4. 10진수 연산 (-25 + 4)를 2의 보수를 이용하여 계산한 결과를 2진수로 표현한 것은? (단, 저장 공간은 8비트로 한다)

① 00010101

② 10010101

③ 11101010

④ 11101011

정답 4번

간단한 해설

-25 + 4는 -21이며, 이를 8비트 **2의 보수**로 나타내면 **11101011**이다. 음수는 절댓값의 이진 표현에서 모든 비트를 반전한 뒤 1을 더해 표현한다.

계산 과정

8비트 2의 보수 표현에서는 양수를 앞에 0을 채워 나타내고, 음수는 **절댓값의 모든 비트를 반전한 뒤 1을 더해** 나타낸다. 이 표현을 사용하면 음수가 포함된 연산도 이진 덧셈으로 처리할 수 있다.

-25의 표현: 25 = 16 + 8 + 1이므로 25의 8비트 이진 표현은 00011001이다. 모든 비트를 반전하면 11100110이고, 여기에 1을 더하면 -25를 나타내는 11100111이 된다.

4의 표현과 덧셈: 양수 4는 00000100이다. 두 수의 같은 자릿수끼리 더하면 11100111 + 00000100 = 11101011이 된다.

결과 해석: 8비트 2의 보수에서 **맨 왼쪽 비트의 가중치는 -128**이다. 따라서 11101011의 값은 -128 + 64 + 32 + 8 + 2 + 1 = -21이며, 주어진 십진수 연산 -25 + 4 = -21과 일치한다. 계산 결과인 11101011은 ④의 값이다.

NoteBase

문 5. 네트워크 시스템을 관리하기 위한 UDP 기반의 프로토콜은?

① SNMP

② SMTP

③ ICMP

④ RARP

정답 1번

간단한 해설

SNMP는 **UDP를 이용해 네트워크 장비의 상태를 감시하고 관리**하는 프로토콜이다.

공통 해설

SNMP(Simple Network Management Protocol)는 네트워크 장비의 관리 정보를 조회하거나 설정하는 응용 계층 프로토콜이다. 관리 시스템의 관리자(manager)가 장비에서 동작하는 에이전트(agent)와 통신하여 장비의 상태 정보를 얻거나 설정을 변경한다. 예를 들어 라우터의 인터페이스별 트래픽 정보를 조회해 통신 상태를 감시할 수 있다. 이러한 관리 메시지를 주고받을 때 일반적으로 **UDP를 사용**하므로, 발문의 네트워크 시스템 관리 목적과 UDP 기반이라는 조건에 대응하는 명칭은 SNMP이다.

NoteBase

문 6. 레지스터에 저장된 10진수 정수 29를 2비트만큼 논리적 우측 시프트(logical right shift)한 결과에 해당하는 10진수의 값은?

① 4

② 7

③ 8

④ 15

정답 2번

간단한 해설

논리적 우측 시프트는 오른쪽으로 밀려난 비트를 버리고 왼쪽 빈자리를 0으로 채운다. 29를 2비트 이동하면 **4로 나눈 정수 몫**인 7이 남는다.

계산 과정

논리적 우측 시프트는 비트열을 오른쪽으로 옮기면서 **왼쪽 빈자리를 0으로 채우고, 오른쪽으로 벗어난 비트를 버리는** 연산이다. 주어진 값은 양의 정수 29이고 이동량은 2비트이다.

$29 = 16 + 8 + 4 + 1$이므로 2진수로는 11101_2이다. 상위의 0을 생략하여 나타내면, 한 비트 우측 시프트한 비트열은 1110_2, 다시 한 비트 우측 시프트한 비트열은 111_2이다. 이 과정에서 원래 값의 **하위 두 비트인 01이 버려진다**.

남은 비트열의 값은 $111_2 = 4 + 2 + 1 = 7$이다. 이는 29를 $2^2 = 4$로 나눈 정수 몫이며, 나머지 1에 해당하는 비트는 시프트 과정에서 사라진다. 따라서 결과의 10진수 값은 7로, ②의 값과 일치한다.

NoteBase

문 7. 다음 설명에 해당하는 소프트웨어 개발 프로세스 모델은?

초기 요구분석 후 프로토타입을 개발하기 이전에 위험 분석 단계를 거친다. 이렇게 만들어진 프로토타입을 사용자가 평가하고, 개발자는 추가 요구사항을 반영하여 다시 위험 분석을 거쳐 2차 프로토타입을 만든다. 이 과정을 반복하면서 사용자의 요구사항이 충분히 반영된 최종 제품을 완성해 나간다.

① 나선형 모델

② 폭포수 모델

③ CBD 모델

④ SPICE 모델

정답 1번

간단한 해설

프로토타입을 만들기 전에 **위험 분석**을 하고, 사용자 평가와 추가 요구사항을 반영해 이 과정을 반복하는 것은 나선형 모델의 특징이다.

공통 해설

나선형 모델은 **위험 분석을 중심으로 개발과 평가를 반복**하여 제품을 점진적으로 완성하는 소프트웨어 개발 프로세스 모델이다. 프로토타입은 사용자가 기능이나 요구사항을 확인할 수 있도록 미리 만든 시제품이다. 지문에서는 초기 요구분석 후 위험 분석을 거쳐 프로토타입을 만들고, 사용자 평가에서 나온 추가 요구사항을 반영한 뒤 다시 위험을 분석하여 다음 프로토타입을 만든다. 앞선 평가 결과가 다음 개발에 반영되므로, 반복할수록 사용자의 요구사항을 더 충실히 반영한 제품으로 발전한다.

NoteBase

문 8. 메모리에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 플래시 메모리는 NAND형과 NOR형이 있다.
ㄴ. SRAM은 DRAM에 비해 접근 속도가 빠른 편이다.
ㄷ. ROM은 휘발성 메모리로 전원 없이 데이터를 유지할 수 없다.
ㄹ. EPROM은 자외선 등을 사용하여 여러 번 저장된 내용을 지우고 다른 내용을 기록할 수 있다.

① ㄱ, ㄴ
② ㄴ, ㄷ
③ ㄷ, ㄹ
④ ㄱ, ㄴ, ㄹ

정답 4번

간단한 해설

ROM은 **전원이 꺼져도 데이터를 유지하는 비휘발성 메모리**이다. EPROM은 ROM 계열이지만 **자외선으로 내용을 지운 뒤 다시 기록**할 수 있다.

보기별 해설

ㄱ. O: 플래시 메모리는 전기적으로 내용을 지우고 다시 기록할 수 있는 비휘발성 메모리이다. 메모리 셀의 연결 구조에 따라 **NAND형과 NOR형**으로 구분한다.

ㄴ. O: SRAM은 플립플롭 구조로 데이터를 저장하여 전원이 공급되는 동안 주기적인 재충전이 필요 없다. DRAM은 축전기에 저장한 전하로 데이터를 나타내며, 전하가 누설되므로 주기적인 재충전이 필요하다. 이러한 저장 구조의 차이로 SRAM은 일반적으로 **DRAM보다 접근 속도가 빠르다**.

ㄷ. X: ROM은 **전원 없이도 데이터를 유지하는 비휘발성 메모리**이다. 따라서 '휘발성 메모리로 전원 없이 데이터를 유지할 수 없다'는 부분은 '비휘발성 메모리로 전원 없이도 데이터를 유지할 수 있다'로 고쳐야 한다.

ㄹ. O: EPROM은 저장된 내용을 지우고 다시 기록할 수 있는 ROM이다. **자외선으로 기존 내용을 지운 뒤** 전기적으로 새 내용을 기록할 수 있으므로, 소거와 기록을 여러 번 반복할 수 있다.

옳은 보기는 ㄱ, ㄴ, ㄹ이며, 이 조합에 해당하는 선택지는 ④이다.

NoteBase

문 9. 모듈의 응집도(cohesion) 및 결합도(coupling)에 대한 설명으로 옳지 않은 것은?

① 한 모듈이 다른 모듈 내부를 직접 참조하는 경우는 내용(content) 결합도에 해당한다.

② 응집도는 하나의 모듈 내에서 구성 요소들이 얼마나 밀접하게 관련되어 있는지를 나타낸다.

③ 모듈을 구성하는 한 요소의 출력이 다음 요소의 입력으로 사용되는 경우는 순차적(sequential) 응집도에 해당한다.

④ 레코드, 구조체 또는 객체와 같은 복잡한 데이터 구조를 모듈 인터페이스를 통해 주고받는 경우는 데이터(data) 결합도에 해당한다.

정답 4번

간단한 해설

④에서 설명한 복잡한 자료구조를 주고받는 방식은 **스탬프(stamp) 결합도**이다. 데이터 결합도는 필요한 개별 데이터 항목을 매개변수로 주고받는 경우이다.

선지별 해설

① O: 내용 결합도는 한 모듈이 **다른 모듈의 내부 코드나 자료를 직접 참조**하는 경우이다. 상대 모듈의 내부 구현에 의존하므로, 그 내부가 바뀌면 참조하는 모듈도 영향을 받을 수 있다.

② O: 응집도는 **한 모듈 내부 구성 요소들 사이의 관련성**을 나타낸다. 구성 요소들이 하나의 목적이나 기능을 위해 밀접하게 연결될수록 응집도가 높다. 서로 다른 모듈 사이의 의존 관계를 나타내는 결합도와 구별한다.

③ O: 순차적 응집도는 한 모듈 안에서 **앞 요소의 출력이 다음 요소의 입력**으로 이어지는 경우이다. 예를 들어 자료를 읽는 부분의 출력이 그 자료를 가공하는 부분의 입력으로 사용되면, 두 부분은 자료의 처리 흐름으로 연결된다.

④ X: 레코드·구조체·객체와 같은 복잡한 자료구조를 모듈 인터페이스로 전달하는 것은 **스탬프(stamp) 결합도**이다. '데이터(data) 결합도'를 '스탬프 결합도'로 고쳐야 한다. 데이터 결합도는 필요한 개별 데이터 항목을 매개변수로 전달하는 경우이다. 예를 들어 이름만 필요한 모듈에 학생 구조체 전체를 전달하면 스탬프 결합도이고, 필요한 이름 값만 전달하면 데이터 결합도이다.

NoteBase

문 10. 다음 C 프로그램의 출력 결과는?

```
#include <stdio.h>

void func(int *p, int *q) {
    if (p > q) return;
    printf("%d ", *p + *q);
    func(p + 1, q - 1);
}

int main(void) {
    int a[] = { 3, 1, 4, 1, 5 };
    func(a, a + 4);
    return 0;
}
```

① 6 2 8

② 8 2 6

③ 8 2 8

④ 8 8 2

정답 3번

간단한 해설

양끝 원소의 합을 출력한 뒤 두 포인터가 안쪽으로 한 원소씩 이동한다. **중앙에서 p와 q가 같아도 출력**하므로 마지막 값은 4 + 4 = 8이며, 출력은 8 2 8이다.

NoteBase

계산 과정

초기 상태: 배열 a의 인덱스 0~4에 저장된 값은 차례로 3, 1, 4, 1, 5이다. func(a, a + 4)에서 p는 a[0]을, q는 a[4]를 가리킨다. *p와 *q는 각각 가리키는 원소의 값이다. 재귀 호출의 p + 1과 q - 1은 두 포인터를 배열 안쪽으로 한 원소씩 이동시킨다.

첫 번째 호출: p가 가리키는 인덱스는 0, q가 가리키는 인덱스는 4이다. p > q가 거짓이므로 *p + *q = 3 + 5 = 8을 출력한다. 이어서 인덱스 1과 3을 가리키는 포인터로 func를 호출한다.

두 번째 호출: p는 a[1]을, q는 a[3]을 가리킨다. p > q가 거짓이므로 1 + 1 = 2를 출력한 뒤, 두 포인터를 각각 인덱스 2로 이동시켜 재귀 호출한다.

세 번째 호출: p와 q가 모두 a[2]를 가리킨다. 종료 조건은 **p > q이며, p == q는 포함하지 않는다**. 따라서 중앙 원소 4를 두 번 더한 4 + 4 = 8을 출력한다.

네 번째 호출과 종료: 다음 호출에서는 p가 인덱스 3, q가 인덱스 1을 가리키므로 p > q가 참이다. 값을 출력하지 않고 반환한다. 앞선 호출들도 재귀 호출 뒤에 출력문이 없으므로 추가 출력 없이 끝난다.

출력 순서는 8 2 8이며, ③의 값과 일치한다.

NoteBase

문 11. 데이터베이스 뷰(view)에 대한 설명으로 옳지 않은 것은?

① 뷰를 이용하여 다른 뷰를 정의할 수 있다.

② 뷰를 제거할 때는 DELETE VIEW 명령을 사용한다.

③ 뷰는 다른 테이블에서 유도된 가상 테이블이다.

④ 뷰를 이용하여 사용자별로 필요한 데이터만 선별하여 보여줄 수 있다.

정답 2번

간단한 해설

뷰 자체를 제거할 때는 **DROP VIEW** 명령을 사용한다. DELETE는 행 데이터를 삭제하는 명령이므로 뷰의 정의를 제거하는 명령과 구별해야 한다.

선지별 해설

① O: 뷰를 정의하는 조회문에서 기본 테이블뿐 아니라 **이미 정의된 다른 뷰를 참조**할 수 있다. 따라서 기존 뷰의 조회 결과에서 필요한 부분을 다시 골라 새로운 뷰를 정의할 수 있다.

② X: 뷰의 정의를 제거하는 명령은 DELETE VIEW가 아니라 **DROP VIEW**이다. DELETE는 행 데이터를 삭제하는 명령이고, DROP VIEW는 데이터베이스 객체인 뷰를 제거하는 명령이다. 뷰를 제거해도 그 뷰가 참조하던 기본 테이블 자체가 제거되지는 않는다.

③ O: 뷰는 기본 테이블이나 다른 뷰에 대한 조회문으로 정의되는 **가상 테이블**이다. 일반적인 뷰는 데이터를 별도로 저장하기보다 조회문의 정의를 저장하며, 사용할 때 그 정의에 따른 결과를 테이블처럼 보여 준다.

④ O: 뷰의 조회문에서 필요한 열을 선택하고 행에 조건을 걸면 **사용자에게 필요한 데이터만** 보여 줄 수 있다. 예를 들어 직원 테이블에서 이름과 부서 열만 선택한 뷰를 제공하면 급여 열을 제외한 결과를 보여 줄 수 있다.

NoteBase

문 12. 2진 소수로 변환하면 무한 순환 소수가 되는 10진 소수는?

① 0.25

② 0.375

③ 0.4

④ 0.5

정답 3번

간단한 해설

0.4를 기약분수로 나타내면 $\frac{2}{5}$이므로 분모에 **2 이외의 소인수 5**가 남는다. 따라서 2진 소수로는 유한하게 끝나지 않고 같은 숫자 배열이 반복된다.

공통 해설

유리수를 기약분수로 나타냈을 때, 분모의 소인수가 **2뿐이면 유한 2진 소수**가 되고 2 이외의 소인수가 있으면 무한 순환 2진 소수가 된다. 소수점 아래가 k자리인 유한 2진 소수는 분모가 2^k인 분수로 나타낼 수 있기 때문이다.

주어진 수 중 $0.25 = \frac{1}{4}$, $0.375 = \frac{3}{8}$, $0.5 = \frac{1}{2}$는 기약분수의 분모가 각각 4, 8, 2이다. 모두 2의 거듭제곱이므로 유한 2진 소수로 표현된다. 반면 $0.4 = \frac{2}{5}$는 기약분수의 분모에 5가 남아 2의 거듭제곱을 분모로 하는 분수로 나타낼 수 없으므로 무한 순환 2진 소수가 된다.

NoteBase

문 13. 다음 릴레이션에서 속성 A ~ D의 함수 종속을 화살표(→)로 표현할 때 옳은 것은? (단, 아래 투플을 기준으로 판단하며, 새로운 투플은 추가되지 않는다고 가정한다)

A	B	C	D
1	2	7	7
2	8	5	6
2	6	5	6
4	8	7	2

① A → B

② B → C

③ {A, D} → B

④ {C, D} → A

정답 4번

간단한 해설

C와 D의 값이 모두 같은 두 투플은 A의 값도 같다. 반복되는 (C, D) = (5, 6)에서 A가 모두 2이므로, 주어진 투플에서는 **C와 D의 조합이 A를 유일하게 결정**한다.

NoteBase

선지별 해설

함수 종속 X → Y는 어떤 두 투플의 X 값이 같으면 Y 값도 반드시 같다는 뜻이다. X가 여러 속성으로 이루어지면 그 속성들의 값이 모두 같은 투플끼리 비교한다. 이 문제는 새로운 투플이 추가되지 않는다는 조건에 따라 표의 네 투플만으로 판단한다.

① X: A가 2인 두 번째와 세 번째 투플의 B는 각각 8과 6이다. **A가 같아도 B가 다르므로** A만으로 B를 유일하게 결정할 수 없으며, A → B는 성립하지 않는다.

② X: B가 8인 두 번째와 네 번째 투플의 C는 각각 5와 7이다. **B가 같아도 C가 다르므로** B만으로 C를 유일하게 결정할 수 없으며, B → C는 성립하지 않는다.

③ X: 두 번째와 세 번째 투플은 (A, D)가 모두 (2, 6)이지만 B는 각각 8과 6이다. **A와 D를 함께 묶어도 B가 하나로 정해지지 않으므로** {A, D} → B는 성립하지 않는다.

④ O: (C, D)가 (5, 6)인 두 번째와 세 번째 투플의 A는 모두 2이다. 나머지 조합인 (7, 7)과 (7, 2)에서는 A가 각각 1과 4로 하나씩 정해진다. 따라서 **C와 D가 모두 같은데 A가 다른 투플 쌍이 없으므로** {C, D} → A가 성립한다.

NoteBase

문 14. FIFO 페이지 교체 알고리즘을 사용하는 가상메모리에서 프로세스 P가 다음과 같은 페이지 번호 순서대로 페이지에 접근할 때, 페이지 부재(page-fault) 발생 횟수는? (단, 프로세스 P가 사용하는 페이지 프레임은 총 3개이고, 빈 상태에서 시작한다)

8 1 2 3 1 4 5 3 4 1 4 3

① 6회

② 7회

③ 8회

④ 9회

정답 3번

간단한 해설

FIFO는 **먼저 적재된 페이지부터 교체**하며, 이미 있는 페이지를 다시 참조해도 적재 순서는 바뀌지 않는다. 빈 프레임에 처음 적재하는 경우까지 세면 페이지 부재는 총 8회 발생한다.

NoteBase

계산 과정

FIFO에서는 요청한 페이지가 프레임에 없으면 페이지 부재가 발생한다. 빈 프레임이 있으면 그곳에 적재하고, 3개가 모두 차 있으면 **가장 먼저 적재된 페이지를 교체**한다. 아래 대괄호는 물리적인 프레임 위치가 아니라, 현재 적재된 페이지를 오래된 것부터 나열한 것이다.

1~3번째 접근 8, 1, 2: 처음에는 프레임이 모두 비어 있으므로 각각 페이지 부재가 발생한다. 적재 순서는 [8, 1, 2]가 되고, 누적 부재는 3회이다.

4번째 접근 3: 3이 없으므로 가장 오래된 8을 교체한다. [1, 2, 3]이 되고, 누적 부재는 4회이다.

5번째 접근 1: 1이 있으므로 부재가 발생하지 않는다. **적중해도 적재 순서는 바뀌지 않으므로** [1, 2, 3]을 유지한다.

6번째 접근 4: 4가 없으므로 가장 오래된 1을 교체한다. 직전에 1을 참조했어도 FIFO의 교체 대상은 달라지지 않는다. [2, 3, 4]가 되고, 누적 부재는 5회이다.

7번째 접근 5: 5가 없으므로 2를 교체한다. [3, 4, 5]가 되고, 누적 부재는 6회이다.

8번째 접근 3: 3이 있으므로 부재가 발생하지 않는다. [3, 4, 5]를 유지한다.

9번째 접근 4: 4가 있으므로 부재가 발생하지 않는다. [3, 4, 5]를 유지한다.

10번째 접근 1: 1이 없으므로 가장 오래된 3을 교체한다. [4, 5, 1]이 되고, 누적 부재는 7회이다.

11번째 접근 4: 4가 있으므로 부재가 발생하지 않는다. [4, 5, 1]을 유지한다.

12번째 접근 3: 3이 없으므로 가장 오래된 4를 교체한다. [5, 1, 3]이 되고, 누적 부재는 8회이다.

페이지 부재는 1·2·3·4·6·7·10·12번째 접근에서 발생하므로 총 8회이며, ③의 8회와 일치한다.

NoteBase

문 15. IPv4 주소의 첫 번째 바이트와 각 바이트에 해당하는 클래스가 옳게 짝지어지지 않은 것은?

① 00010011 — Class C

② 01001010 — Class A

③ 10010011 — Class B

④ 11100110 — Class D

정답 1번

간단한 해설

00010011은 첫 비트가 **0**이므로 Class A에 해당한다. Class C는 첫 세 비트가 **110**인 주소이다.

선지별 해설

IPv4의 클래스는 첫 번째 바이트를 왼쪽의 최상위 비트부터 읽어 구분한다. Class A는 0, Class B는 10, Class C는 110, Class D는 1110으로 시작한다.

① X: 00010011은 첫 비트가 0이므로 **Class A**에 해당한다. Class C는 첫 세 비트가 110이어야 하므로, 'Class C'를 'Class A'로 고쳐야 한다.

② O: 01001010은 첫 비트가 **0**이므로 Class A에 해당한다. 그다음 비트가 1이어도 첫 비트가 0이라는 Class A의 구분 조건을 만족한다.

③ O: 10010011은 첫 두 비트가 **10**이므로 Class B에 해당한다. Class B는 이 두 비트로 구분한다.

④ O: 11100110은 첫 네 비트가 **1110**이므로 Class D에 해당한다. 첫 네 비트가 1111인 Class E와 구별된다.

NoteBase

문 16. 다음 Java 프로그램의 출력 결과는?

```
class ClassA {
    ClassA() {
        System.out.print("1");
    }
    static {
       System.out.print("4");
    }
}
public class ClassB extends ClassA {
    ClassB() {
        System.out.print("2");
    }
    public static void main(String[] args) {
        System.out.print("3");
        new ClassB();
    }
}
```

① 32

② 432

③ 3214

④ 4312

정답 4번

간단한 해설

main() 실행 전에 상위 클래스의 static 초기화 블록이 실행되고, 객체를 생성할 때는 **상위 클래스 생성자부터** 실행된다. 따라서 4, 3, 1, 2가 차례로 출력되어 4312가 된다.

NoteBase

계산 과정

ClassA의 static 초기화 블록: 프로그램 시작 시 main()을 실행할 ClassB가 초기화되기 전에 **상위 클래스 ClassA가 먼저 초기화**된다. 이때 ClassA의 static 블록이 실행되어 4를 출력한다. 소스 코드에서 생성자가 static 블록보다 위에 있어도, 생성자는 객체를 만들 때 실행되므로 이 시점에는 1이 출력되지 않는다.

main() 실행: 클래스 초기화를 마친 뒤 ClassB의 main() 본문을 실행한다. 첫 문장인 System.out.print("3")이 실행되어 출력은 43이 된다. 다음 문장인 new ClassB()에서 객체 생성이 시작된다.

ClassA() 실행: ClassB()에는 다른 생성자를 호출하는 문장이 없으므로 **상위 클래스의 기본 생성자를 호출하는 super()가 암묵적으로 적용**된다. 이에 따라 ClassB() 본문의 출력문보다 ClassA() 본문의 출력문이 먼저 실행되어 1이 추가된다. 이 시점의 출력은 431이다.

ClassB() 실행: ClassA()의 실행이 끝나면 ClassB() 본문의 System.out.print("2")가 실행되어 2가 추가된다. 이미 초기화된 ClassA의 static 블록은 객체를 생성한다고 다시 실행되지 않는다.

각 출력문은 줄바꿈이나 공백을 넣지 않는 print()를 사용하므로 최종 출력은 4312이며, ④의 값과 일치한다.

NoteBase

문 17. 교착상태(deadlock)에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 교착상태 발생의 필요조건은 상호 배제, 선점, 점유와 대기, 선형 대기이다.
ㄴ. 프로세스를 강제로 종료해서 교착상태 문제를 처리하는 것은 교착상태 회복 기법에 해당한다.
ㄷ. 어떤 제약을 가하지 않고 자원 할당 그래프를 지속적으로 모니터링하면서 교착상태 발생 여부를 살펴보는 것은 교착상태 예방 기법에 해당한다.
ㄹ. 은행원 알고리즘(banker's algorithm) 등을 이용하여 안전(safe) 상태를 유지할 수 있는 범위 내에서 자원을 할당하는 것은 교착상태 회피 기법에 해당한다.

① ㄱ, ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄹ
④ ㄷ, ㄹ

정답 3번

간단한 해설

교착상태가 발생한 뒤 프로세스를 강제로 종료해 자원을 회수하는 것은 **회복**이다. 은행원 알고리즘은 자원을 할당하기 전에 안전 상태를 유지할 수 있는지 확인하는 **회피** 기법이다.

NoteBase

보기별 해설

ㄱ. X: 교착상태의 네 가지 필요조건은 상호 배제, 비선점, 점유와 대기, 환형 대기이다. 보기의 '선점'과 '선형 대기'는 각각 **비선점과 환형 대기**로 고쳐야 한다. 상호 배제는 한 자원을 한 번에 한 프로세스만 사용할 수 있다는 뜻이고, 비선점은 사용 중인 자원을 다른 프로세스가 강제로 빼앗을 수 없다는 뜻이다. 점유와 대기는 자원을 가진 채 다른 자원을 기다리는 것이며, 환형 대기는 프로세스들이 원형으로 연결되어 다음 프로세스가 가진 자원을 기다리는 것이다.

ㄴ. O: 교착상태 회복은 **이미 발생한 교착상태를 해소**하는 방법이다. 관련 프로세스를 모두 종료하거나 일부를 차례로 종료하면, 그 프로세스가 점유하던 자원이 반환되어 다른 프로세스의 대기를 풀 수 있다. 따라서 프로세스 강제 종료는 회복 기법에 해당한다.

ㄷ. X: 자원 할당에 사전 제약을 가하지 않고 자원 할당 그래프를 살펴 교착상태 발생 여부를 확인하는 것은 **교착상태 탐지** 기법이다. 예방은 네 가지 필요조건 중 적어도 하나가 성립하지 않도록 자원 요청이나 할당 방식에 제약을 두는 방법이다. 예를 들어 자원 요청 순서를 정해 그 순서대로만 요청하게 하면 환형 대기를 막을 수 있다.

ㄹ. O: 교착상태 회피는 자원 요청을 허용한 뒤에도 **안전 상태가 유지되는지** 확인하여 할당 여부를 결정한다. 안전 상태란 각 프로세스가 앞으로 요구할 수 있는 자원까지 고려하더라도 모든 프로세스를 차례로 완료시킬 수 있는 실행 순서가 존재하는 상태이다. 은행원 알고리즘은 프로세스별 최대 자원 요구량과 현재 할당량 등을 이용해 이를 검사하고, 안전 상태를 유지하는 요청만 허용한다.

옳은 보기는 ㄴ과 ㄹ이며, 이 조합은 ③에 해당한다.

NoteBase

문 18. 다음 파이썬 코드로 작성된 partition() 함수를 이용하여, 주어진 배열을 퀵 정렬(quick sort)로 오름차순 정렬하고자 한다. 정렬 과정에서 단계별 정렬 순서로 나타날 수 없는 것은? (단, 피벗(pivot)은 정렬하고자 하는 대상의 마지막 원소로 선택한다)

```
# 정렬하고자 하는 대상인 A[first]…A[last]를
# 피벗(A[last]) 기준으로 분할하는 함수
def partition(A, first, last):
    p = A[last]
    low = first
    high = last
    while low < high:
        while p > A[low] and low < high:
            low += 1
        while p <= A[high] and low < high:
            high -= 1
        if low < high:
            A[low], A[high] = A[high], A[low]
    A[low], A[last] = A[last], A[low]
    return low
```

배열	7	3	2	19	13	5	11	17

① 2, 3, 5, 7, 13, 11, 17, 19

② 2, 3, 5, 11, 7, 13, 17, 19

③ 2, 3, 5, 11, 13, 7, 17, 19

④ 7, 3, 2, 11, 13, 5, 17, 19

정답 2번

간단한 해설

부분 배열 [11, 13, 7]에서는 피벗 7이 가운데 13과 교환되는 것이 아니라 **첫 원소 11과 직접 교환**된다. 따라서 [7, 13, 11]이 되며, [11, 7, 13]인 상태는 나타나지 않는다.

NoteBase

공통 해설

주어진 partition()은 대상 구간의 마지막 원소를 피벗 p로 정한다. low는 왼쪽에서 p 이상인 원소를 만날 때까지, high는 오른쪽에서 p 미만인 원소를 만날 때까지 이동한다. 두 위치가 만나기 전에는 해당 원소끼리 교환하고 탐색을 계속한다. 두 위치가 만나면 **A[low]와 마지막 원소인 피벗을 교환**하여 피벗의 위치를 확정한다. 따라서 피벗보다 작은 값과 큰 값이 양쪽에 나뉘어 있다는 사실만으로는 충분하지 않고, 이 코드의 교환으로 실제 만들어지는 배열인지 확인해야 한다.

처음 배열 [7, 3, 2, 19, 13, 5, 11, 17]의 피벗은 17이다. low는 19에서, high는 11에서 멈추므로 두 값을 교환한다. 이후 low와 high가 19의 위치에서 만나고, 그 19와 피벗 17을 교환한다. 분할 결과는 [7, 3, 2, 11, 13, 5, 17, 19]로, ④의 배열이 나타난다.

17 왼쪽 구간 [7, 3, 2, 11, 13, 5]의 피벗은 5이다. 7과 2를 먼저 교환하면 [2, 3, 7, 11, 13, 5]가 된다. 이어 low와 high가 7의 위치에서 만나므로 7과 피벗 5를 교환한다. 전체 배열은 [2, 3, 5, 11, 13, 7, 17, 19]가 되어 ③의 배열이 나타난다. 왼쪽의 [2, 3]을 분할해도 원소의 순서는 달라지지 않는다.

5 오른쪽 구간 [11, 13, 7]의 피벗은 7이다. low는 첫 원소 11에서 멈추고, high는 7과 13을 지나 low의 위치까지 이동한다. 두 위치가 만났으므로 반복문 안의 교환은 실행되지 않고, **마지막 교환에서 11과 피벗 7을 맞바꾼다**. 그 결과는 [7, 13, 11]이며, 전체 배열 [2, 3, 5, 7, 13, 11, 17, 19]는 ①과 같다. 이후 [13, 11]을 분할하면 [11, 13]이 되어 정렬이 끝난다. 이 과정에서 [11, 7, 13]이 만들어지는 교환은 없으므로, [2, 3, 5, 11, 7, 13, 17, 19]인 상태는 나타날 수 없다.

NoteBase

문 19. 디스크 스케줄링 기법 중 SSTF를 사용하여 다음과 같은 작업 대기 큐(디스크 큐)의 작업을 처리하고자 하는 경우, 처리되는 트랙의 순서를 바르게 나열한 것은? (단, 추가 I/O 요청은 없다고 가정한다)

◯ 작업 대기 큐: 15, 4, 8, 120, 65, 70, 105
◯ 헤드 시작 위치: 50

① 65 → 70 → 105 → 120 → 15 → 8 → 4
② 65 → 70 → 120 → 105 → 15 → 8 → 4
③ 70 → 65 → 105 → 120 → 15 → 8 → 4
④ 70 → 65 → 120 → 15 → 8 → 4 → 105

정답 1번

간단한 해설

SSTF는 **현재 헤드 위치에서 가장 가까운 미처리 요청**을 매번 선택한다. 요청 하나를 처리할 때마다 기준 위치가 바뀌므로, 70에서는 거리가 50인 120보다 거리가 35인 105를 먼저 처리한다.

NoteBase

올바른 순서

65 → 70 → 105 → 120 → 15 → 8 → 4

65: SSTF는 현재 헤드에서 이동 거리가 가장 짧은 미처리 요청을 선택한다. 시작 위치 50에서 65까지는 15트랙, 70까지는 20트랙이고, 나머지 요청까지의 거리는 모두 35트랙 이상이므로 65를 먼저 처리한다.

70: 65를 처리하면 **헤드 위치가 65로 바뀐다**. 남은 요청 중 70까지의 거리가 5트랙으로 가장 짧으므로 70을 처리한다.

105: 현재 위치 70에서 105까지는 35트랙, 120까지는 50트랙, 15까지는 55트랙이다. 8과 4는 더 멀리 있으므로 105를 선택한다. 대기 큐에 먼저 적힌 요청이 아니라 **현재 위치에서 더 가까운 요청**이 우선한다.

120: 현재 위치 105에서 120까지는 15트랙이다. 남은 15, 8, 4까지는 각각 90, 97, 101트랙이므로 120을 처리한다.

15: 현재 위치 120에서 남은 요청 15, 8, 4까지의 거리는 각각 105, 112, 116트랙이다. 이 중 가장 가까운 15를 처리하며, 헤드는 트랙 번호가 감소하는 방향으로 이동한다.

8: 현재 위치 15에서 8까지는 7트랙, 4까지는 11트랙이므로 8을 먼저 처리한다.

4: 현재 위치 8에서 남은 요청은 4뿐이다. 4트랙을 이동하여 이를 처리하면, 추가 I/O 요청이 없다는 조건에 따라 모든 요청의 처리가 끝난다.

NoteBase

문 20. 다음 C 프로그램의 출력 결과는?

```c
#include <stdio.h>

void rotate(int *a, int n, int k) {
    k %= n;
    while (k--) {
        int first = a[0];
        int i;
        for (i = 0; i < n - 1; i++) {
            a[i] = a[i + 1];
        }
        a[n - 1] = first;
    }
}

int main(void) {
    int arr[] = {1, 2, 3, 4, 5};
    rotate(arr, 5, 7);
    int i;
    for (i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

① 1 2 3 4 5

② 3 4 5 1 2

③ 4 5 1 2 3

④ 5 1 2 3 4

정답 2번

간단한 해설

k %= n에 따라 이동 횟수는 7을 5로 나눈 나머지인 2가 된다. 반복할 때마다 첫 원소를 맨 뒤로 보내는 **왼쪽 순환 이동**을 하므로 출력은 3 4 5 1 2이다.

NoteBase

계산 과정

초기 조건: arr의 원소는 1 2 3 4 5이며, rotate(arr, 5, 7)을 호출하므로 n = 5, k = 7이다. 포인터 a는 arr의 원소를 가리키므로 함수 안에서 a의 원소를 바꾸면 원래 배열 arr에도 반영된다.

반복 횟수: k %= n은 k를 n으로 나눈 나머지로 갱신한다. 7 % 5 = 2이므로 k는 2가 된다. 원소가 5개인 배열을 5번 순환 이동하면 처음 배열로 돌아오기 때문에 나머지 횟수만 이동해도 된다. while (k--)는 **감소 전 값으로 조건을 판단**하므로, 조건에서 2와 1을 사용할 때 본문을 실행하고 0을 사용할 때 종료한다. 따라서 본문은 두 번 실행된다.

한 번의 이동: 먼저 a[0]을 first에 보관한다. 이어서 i가 0부터 3까지 증가하며 a[i] = a[i + 1]을 실행하므로 뒤쪽 원소가 한 칸씩 앞으로 이동한다. 마지막으로 a[n - 1] = first를 실행하여 **보관한 첫 원소를 맨 뒤에 배치**한다. 즉, 한 번 실행할 때마다 배열 전체가 왼쪽으로 한 칸 순환 이동한다.

배열 변화: 첫 번째 반복에서는 1을 보관하고 나머지를 앞으로 옮긴 뒤 1을 맨 뒤에 넣으므로 2 3 4 5 1이 된다. 두 번째 반복에서는 이 배열의 첫 원소인 2를 맨 뒤로 보내므로 3 4 5 1 2가 된다.

출력: main의 for문은 인덱스 0부터 4까지 arr[i]를 차례대로 출력한다. 따라서 출력 결과는 3 4 5 1 2이며, ②와 일치한다.

NoteBase