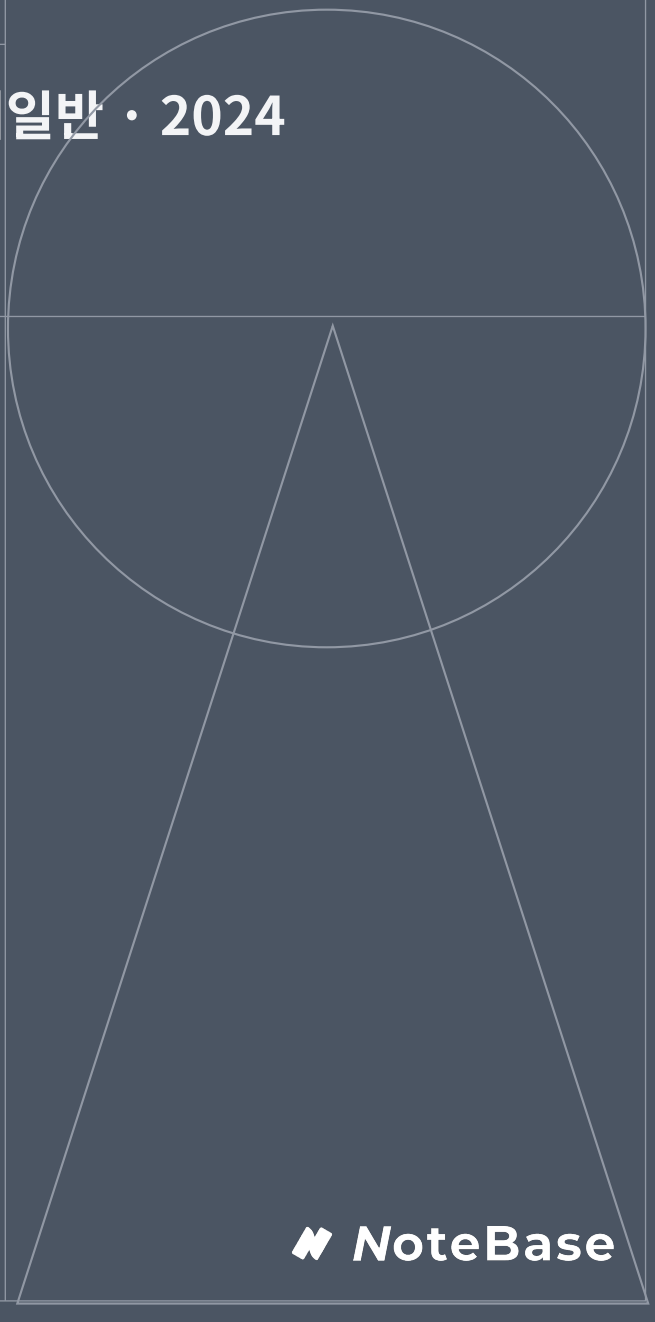
기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2024



NoteBase

기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2024

공무원/국가직/9급 · 컴퓨터일반 · 2024

문 1. 컴퓨터에서 사용하는 정보량의 단위를 크기가 작은 것부터 큰 것 순서대로 바르게 나열한 것은?

① EB, GB, PB, TB

② EB, PB, GB, TB

③ GB, TB, EB, PB

④ GB, TB, PB, EB

정답 4번

간단한 해설

정보량 단위는 바이트 앞에 붙은 접두어의 크기로 비교한다. **페타(P)는 테라(T)보다 크고 엑사(E)보다 작다**.

올바른 순서

GB → TB → PB → EB

GB(기가바이트): 십진 접두어 기준으로 10^9바이트를 뜻한다. 제시된 네 단위 중 가장 작으므로 처음에 놓인다.

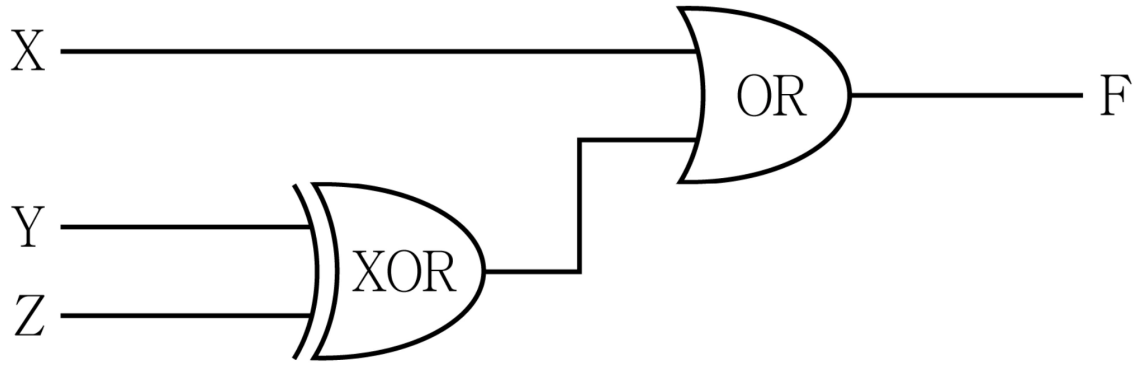
TB(테라바이트): 10^{12}바이트로, **GB의 1,000배**이다. 따라서 GB 다음에 놓인다.

PB(페타바이트): 10^{15}바이트로, **TB의 1,000배**이다. 따라서 TB보다 뒤에 놓인다.

EB(엑사바이트): 10^{18}바이트로, **PB의 1,000배**이다. 제시된 네 단위 중 가장 크므로 마지막에 놓인다.

NoteBase

문 2. 다음 논리회로도에서 출력 F가 0이 되는 입력 조합을 바르게 연결한 것은?



① X: 0, Y: 0, Z: 1

② X: 0, Y: 1, Z: 0

③ X: 0, Y: 1, Z: 1

④ X: 1, Y: 0, Z: 0

정답 3번

간단한 해설

OR 게이트는 두 입력이 모두 0일 때만 0을 출력하고, XOR 게이트는 두 입력이 같을 때 0을 출력한다. 따라서 **X = 0이고 Y = Z**이면 최종 출력 F = 0이 된다.

계산 과정

회로에서 Y와 Z는 먼저 XOR 게이트에 입력되고, 그 출력이 X와 함께 OR 게이트에 입력된다. 따라서 출력은 F = X (Y Z)로 나타낸다. 여기서 는 OR, ⊕는 XOR 연산이다.

OR 출력의 조건: OR 게이트는 입력 중 하나라도 1이면 1을 출력한다. 따라서 F가 0이 되려면 **X와 XOR 출력이 모두 0**이어야 한다.

XOR 출력의 조건: XOR 게이트는 두 입력이 서로 다르면 1, 같으면 0을 출력한다. 따라서 XOR 출력이 0이 되려면 **Y와 Z가 모두 0이거나 모두 1**이어야 한다.

입력 대입: ③의 X = 0, Y = 1, Z = 1을 대입하면 XOR 출력은 $1 \oplus 1 = 0$이다. 이 중간값과 X = 0이 OR 게이트에 입력되므로 최종 출력은 F = 0 0 = 0이 된다.

NoteBase

문 3. 암호화 및 복호화를 위하여 개인키와 공개키가 필요한 비대칭키 암호화 기법은?

① AES

② DES

③ RSA

④ SEED

정답 3번

간단한 해설

RSA는 서로 짝을 이루는 **공개키와 개인키**를 사용하는 비대칭키 암호화 기법이다. 공개키로 암호화한 내용은 그에 대응하는 개인키로 복호화한다.

공통 해설

비대칭키 암호화는 **암호화 키와 복호화 키가 서로 다른** 방식으로, 수학적으로 연관된 공개키와 개인키를 한 쌍으로 사용한다. RSA로 비밀 메시지를 보낼 때는 송신자가 수신자의 공개키로 암호화하고, 수신자는 자신만 보관하는 개인키로 복호화한다. 따라서 공개키를 다른 사람에게 알려 주어도 개인키는 비밀로 유지한다.

RSA는 이러한 공개키·개인키 쌍을 사용하는 반면, AES·DES·SEED는 **암호화와 복호화에 같은 비밀키**를 사용하는 대칭키 암호화 기법이다. 이 문제에서 요구하는 구별 기준은 공개키와 개인키를 나누어 사용하는지 여부이다.

NoteBase

문 4. OSI 모형의 네트워크 계층 프로토콜에 속하지 않는 것은?

① ICMP

② IGMP

③ IP

④ SLIP

정답 4번

간단한 해설

SLIP는 직렬 회선에서 IP 패킷을 전송하기 위한 **데이터 링크 계층** 프로토콜이다. IP 패킷을 운반한다는 이유로 네트워크 계층에 분류하지 않는다.

개념 구분

IP·ICMP·IGMP: 네트워크 계층에서 IP는 논리 주소인 IP 주소를 바탕으로 패킷을 목적지까지 전달한다. ICMP는 IP 통신의 오류 보고와 진단을, IGMP는 IP 멀티캐스트 그룹의 구성원 관리를 담당한다. 세 프로토콜은 역할이 다르지만 모두 **네트워크 계층**에 속한다.

SLIP: 직렬 회선으로 IP 패킷을 보낼 때 수신 측이 패킷의 시작과 끝을 구분할 수 있도록 경계를 표시하는 프레이밍 기능을 제공한다. 이는 **하나의 링크에서 패킷을 운반하는** 데이터 링크 계층의 역할이다. 여러 네트워크를 거쳐 목적지까지 패킷을 전달하는 IP와는 담당 계층이 다르다.

NoteBase

문 5. 클라우드 컴퓨팅 서비스에서 애플리케이션을 구축, 테스트, 설치할 수 있도록 통합환경을 제공하는 것은?

① IaaS

② NAS

③ PaaS

④ SaaS

정답 3번

간단한 해설

애플리케이션을 구축·테스트·설치하는 데 필요한 **개발·실행 플랫폼**을 클라우드 서비스로 제공하는 방식이 PaaS이다.

공통 해설

PaaS(Platform as a Service)는 애플리케이션 개발과 실행에 필요한 플랫폼을 클라우드에서 제공하는 서비스이다. 서비스 제공자가 운영체제, 실행환경, 개발 도구 등의 기반을 마련하므로, 이용자는 이를 활용해 자신의 애플리케이션을 개발하고 테스트한 뒤 설치·배포할 수 있다.

문항의 통합환경은 **애플리케이션을 만드는 과정부터 실행할 수 있도록 배치하는 과정까지 지원하는 환경**을 뜻한다. 이처럼 개발자가 애플리케이션을 구축·테스트·설치할 수 있는 플랫폼을 제공한다는 특징이 PaaS에 해당한다.

NoteBase

문 6. 10진수 뺄셈 (7 - 12)를 2의 보수를 이용하여 계산한 결과는? (단, 저장 공간은 8비트로 한다)

① 0000 0100

② 0000 0101

③ 1111 0101

④ 1111 1011

정답 4번

간단한 해설

7 - 12는 7에 **-12의 2의 보수 표현**을 더해 계산한다. 8비트에서 -12는 1111 0100이므로, 합은 -5를 나타내는 1111 1011이다.

계산 과정

8비트 표현: 10진수 7은 0000 0111, 12는 0000 1100이다. 7 - 12를 계산하려면 7에 -12를 더하면 되므로, 먼저 12의 2의 보수를 구한다.

-12 만들기: 0000 1100의 **각 비트를 반전한 뒤 1을 더한다**. 반전하면 1111 0011이고, 여기에 1을 더하면 1111 0100이다. 이것이 8비트 2의 보수 방식으로 표현한 -12이다.

덧셈: 7의 비트열과 -12의 비트열을 2진수 덧셈으로 더한다.
0000 0111 + 1111 0100 = 1111 1011

결과 해석: 1111 1011은 최상위 비트가 1이므로 2의 보수 방식에서 음수이다. 이 비트열을 반전하고 1을 더하면 0000 0101, 즉 10진수 5가 되므로 나타내는 값은 -5이다. 따라서 계산 결과 1111 1011은 ④와 일치한다.

NoteBase

문 7. RAID(Redundant Array of Inexpensive Disks) 레벨에 대한 설명으로 옳지 않은 것은?

① RAID 레벨 0: 패리티 없이 데이터를 분산 저장한다.

② RAID 레벨 1: 패리티 비트를 사용하여 오류를 검출한다.

③ RAID 레벨 2: 해밍 코드를 사용하여 오류 검출 및 정정이 가능하다.

④ RAID 레벨 5: 데이터와 함께 패리티 정보를 블록 단위로 분산 저장한다.

정답 2번

간단한 해설

RAID 1은 패리티 비트 대신 **미러링**을 사용한다. 동일한 데이터를 다른 디스크에도 복제하여 저장하므로, 한 디스크가 고장 나면 다른 디스크의 복사본을 이용할 수 있다.

선지별 해설

① O: RAID 0은 데이터를 나누어 여러 디스크에 저장하는 스트라이핑 방식이다. **패리티나 복제본을 저장하지 않으므로** 여러 디스크를 병렬로 이용할 수 있지만, 디스크 고장에 대비한 데이터 복구 기능은 제공하지 않는다.

② X: RAID 1의 데이터 보호 방식은 패리티가 아니라 **동일한 데이터를 중복 저장하는 미러링**이다. 예를 들어 두 디스크에 같은 내용을 저장하면, 한 디스크가 고장 나더라도 다른 디스크의 복사본으로 데이터를 이용할 수 있다.

③ O: RAID 2는 데이터를 비트 단위로 나누어 저장하고, **해밍 코드**를 위한 검사 비트도 별도 디스크에 저장한다. 해밍 코드는 검사 비트의 관계를 이용하여 오류를 검출하고 정정할 수 있게 한다.

④ O: RAID 5는 데이터를 블록 단위로 나누어 저장하며, 패리티 정보도 **특정 디스크에 집중하지 않고 여러 디스크에 분산**하여 저장한다. 디스크 하나가 고장 나면 남은 데이터와 패리티 정보를 이용하여 손실된 데이터를 복구할 수 있다.

NoteBase

문 8. RISC와 비교하여 CISC의 특징으로 옳지 않은 것은?

① 명령어의 종류가 많다.

② 명령어의 길이가 고정적이다.

③ 명령어 파이프라인이 비효율적이다.

④ 회로 구성이 복잡하다.

정답 2번

간단한 해설

CISC는 명령어에 따라 필요한 정보의 양이 달라 **가변 길이 명령어**를 사용하는 것이 전형적인 특징이다. 명령어 길이가 고정적이라는 설명은 전통적인 RISC의 특징에 해당한다.

선지별 해설

① O: CISC는 복잡한 작업도 하나의 명령어로 표현할 수 있도록 **다양하고 복잡한 명령어**를 제공한다. 단순한 명령어 위주로 구성하는 RISC와 비교하면 명령어의 종류가 많다.

② X: CISC는 명령어의 기능과 피연산자 지정 방식 등에 따라 **명령어의 길이가 달라지는 가변 길이 방식**을 사용한다. 전통적인 RISC는 명령어 길이를 고정하여 명령어 인출과 해독을 단순하게 한다.

③ O: 파이프라인은 여러 명령어의 인출·해독·실행 등의 단계를 겹쳐 처리하는 방식이다. 전통적인 CISC는 명령어의 길이와 실행 시간이 다양하여, RISC에 비해 **처리 단계를 일정하게 맞추기 어렵다**. 따라서 단순하고 규칙적인 파이프라인을 구성하는 데 불리하다.

④ O: CISC는 다양한 명령어 형식과 주소 지정 방식을 지원하므로, 이를 **해독하고 실행을 제어하는 회로**가 복잡해진다. 단순하고 규칙적인 명령어를 사용하는 RISC보다 제어 회로 구성이 복잡하다는 전통적인 비교이다.

NoteBase

문 9. 다음 파이썬 코드는 이진 탐색을 이용하여 자연수 데이터를 탐색하는 함수이다. (가), (나)에 들어갈 내용을 바르게 연결한 것은? (단, ds는 오름차순으로 정렬된 중복 없는 자연수 리스트이고, key는 찾고자 하는 값이다)

```
def binary(ds, key):
    low = 0
    high = len(ds) - 1
    while low <= high:
        mid = (low+high) // 2
        if key == ds[mid]:
            return mid
        elif key < ds[mid]:
            (가)
        else:
            (나)
    return
```

① (가): high = mid - 1, (나): low = mid - 1

② (가): high = mid - 1, (나): low = mid + 1

③ (가): high = mid + 1, (나): low = mid - 1

④ (가): high = mid + 1, (나): low = mid + 1

정답 2번

간단한 해설

오름차순 리스트에서 key가 중앙값보다 작으면 **왼쪽 구간**을, 크면 **오른쪽 구간**을 탐색한다. 이미 비교하여 일치하지 않음을 확인한 중앙 위치는 다음 탐색 범위에서 제외한다.

NoteBase

빈칸별 해설

low와 high는 현재 탐색 범위의 시작 인덱스와 끝 인덱스이며, 두 끝 위치를 모두 포함한다. mid는 그 범위의 중앙 인덱스이다. 중앙값이 key와 같으면 mid를 반환하고, 다르면 정렬 순서에 따라 key가 있을 수 있는 쪽으로 범위를 줄인다.

(가): high = mid - 1. key < ds[mid]이면 오름차순 정렬 조건상 중앙 위치와 그 오른쪽의 값은 모두 key보다 크다. 따라서 **mid 바로 앞을 탐색 범위의 끝**으로 삼아 왼쪽 구간만 남긴다. high = mid + 1로 바꾸면 제외해야 할 중앙 위치와 오른쪽 위치가 범위에 남을 수 있다.

(나): low = mid + 1. 앞의 두 조건에서 같은 경우와 작은 경우를 처리했으므로 else에서는 key > ds[mid]이다. 중앙 위치와 그 왼쪽의 값은 모두 key보다 작으므로 **mid 바로 뒤를 탐색 범위의 시작**으로 삼는다. low = mid - 1로 바꾸면 제외해야 할 중앙 위치와 왼쪽 위치가 범위에 남을 수 있다.

따라서 (가)에 high = mid - 1, (나)에 low = mid + 1을 넣는 조합이 ②와 일치한다.

NoteBase

문 10. 3개의 페이지 프레임으로 구성된 기억장치에서 다음과 같은 참조열 순으로 페이지가 참조될 때, 페이지 부재 발생 횟수가 가장 적은 교체 방법은? (단, 초기 페이지 프레임은 비어 있으며, 페이지 교체 과정에서 사용 빈도수가 동일한 경우는 가장 오래된 것을 먼저 교체한다)

참조열: 2 1 2 3 1 4 5 1 4 3

① FIFO(First In First Out)

② LFU(Least Frequently Used)

③ LRU(Least Recently Used)

④ MFU(Most Frequently Used)

정답 3번

간단한 해설

LRU는 **마지막 참조 시점이 가장 오래된 페이지**를 교체한다. 이 참조열에서는 페이지 1과 4를 유지하여 8·9번째 참조가 모두 적중하므로, 페이지 부재가 **6회**로 가장 적다.

NoteBase

선지별 해설

참조한 페이지가 프레임에 없으면 페이지 부재가 발생한다. 초기 프레임은 비어 있으므로 처음 페이지를 적재할 때도 부재로 센다. 참조열 2, 1, 2, 3, 1, 4, 5, 1, 4, 3에 차례로 1~10번째 번호를 붙여 비교한다. 사용 빈도가 같은 경우에는 먼저 적재된 페이지를 교체한다. 아래 교체 과정에서 '2→4'는 페이지 2를 내보내고 페이지 4를 적재한다는 뜻이다.

① FIFO: **가장 먼저 적재된 페이지**를 교체하며, 적중이 발생해도 적재 순서는 바뀌지 않는다. 프레임이 찬 뒤의 교체는 6번째 2→4, 7번째 1→5, 8번째 3→1, 10번째 4→3이다. 페이지 부재는 1·2·4·6·7·8·10번째 참조에서 발생하여 총 7회이다.

② LFU: **참조 횟수가 가장 적은 페이지**를 교체한다. 6번째 참조 직전에는 페이지 2와 1이 각각 2회, 페이지 3이 1회 참조되었으므로 3→4로 교체한다. 이후에는 참조 횟수가 1회인 페이지를 차례로 내보내어 7번째 4→5, 9번째 5→4, 10번째 4→3으로 교체한다. 페이지 부재는 1·2·4·6·7·9·10번째 참조에서 발생하여 총 7회이다.

③ LRU: **마지막으로 참조된 시점이 가장 오래된 페이지**를 교체한다. 6번째 참조 직전에는 페이지 2·3·1이 각각 3·4·5번째에 마지막으로 참조되었으므로 2→4로 교체한다. 7번째에는 가장 오래 참조되지 않은 페이지 3을 내보내어 3→5로 교체한다. 이때 남은 페이지는 1·4·5이므로 8번째의 페이지 1과 9번째의 페이지 4는 모두 적중한다. 10번째에는 5→3으로 교체한다. 페이지 부재는 1·2·4·6·7·10번째 참조에서 발생하여 총 6회이다.

④ MFU: **참조 횟수가 가장 많은 페이지**를 교체한다. 6번째에는 페이지 2와 1의 참조 횟수가 모두 2회이므로, 먼저 적재된 페이지 2를 내보내어 2→4로 교체한다. 7번째에는 참조 횟수가 2회인 페이지 1을 내보내어 1→5로 교체한다. 8번째에는 페이지 3·4·5의 참조 횟수가 모두 1회이므로 가장 먼저 적재된 페이지 3을 내보내어 3→1로 교체한다. 9번째에 페이지 4가 적중하여 참조 횟수가 2회가 되므로, 10번째에는 4→3으로 교체한다. 페이지 부재는 1·2·4·6·7·8·10번째 참조에서 발생하여 총 7회이다.

같은 3개 프레임과 참조열에서 FIFO·LFU·MFU는 각각 7회, LRU는 6회이다. 따라서 페이지 부재 발생 횟수가 가장 적은 방법은 ③ LRU이다.

NoteBase

문 11. 교착상태(deadlock)가 발생하기 위한 필요조건에 해당하지 않는 것은?

① 상호 배제(mutual exclusion)

② 선점(preemption)

③ 순환 대기(circular wait)

④ 점유와 대기(hold and wait)

정답 2번

간단한 해설

교착상태의 필요조건은 선점이 아니라 **비선점**이다. 비선점은 다른 프로세스가 사용 중인 자원을 강제로 빼앗을 수 없다는 뜻이다.

개념 구분

교착상태는 프로세스들이 서로 상대가 보유한 자원을 기다려 더 이상 진행하지 못하는 상태이다. 발생하려면 상호 배제, 점유와 대기, 비선점, 순환 대기의 네 조건이 모두 성립해야 한다.

상호 배제: 어떤 자원은 **한 번에 하나의 프로세스만** 사용할 수 있다. 다른 프로세스가 그 자원을 사용하려면 기다려야 한다.

점유와 대기: 프로세스가 이미 자원을 **보유한 채 다른 자원을 기다리는** 조건이다. 기다리는 동안에도 기존 자원을 놓지 않는다.

순환 대기: 프로세스들의 자원 대기가 고리를 이루는 조건이다. 예를 들어 A가 B의 자원을 기다리고, B도 A의 자원을 기다리는 관계이다.

비선점과 선점: 비선점은 프로세스가 자원을 스스로 반납할 때까지 **강제로 회수할 수 없는** 조건이다. 반대로 선점은 사용 중인 자원을 강제로 회수하는 것이므로 교착상태의 필요조건에 해당하지 않는다. 여기서 구별하는 것은 CPU 스케줄링 방식이 아니라 자원의 강제 회수 가능 여부이다.

NoteBase

문 12. 다음 CPU 스케줄링 알고리즘 중 비선점형 알고리즘만을 모두 고르면?

ㄱ. FCFS(First Come First Served) 스케줄링
ㄴ. HRN(Highest Response-ratio Next) 스케줄링
ㄷ. RR(Round Robin) 스케줄링
ㄹ. SRT(Shortest Remaining Time) 스케줄링

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄷ, ㄹ

정답 1번

간단한 해설

FCFS와 HRN은 실행 중인 프로세스의 CPU를 다른 프로세스가 강제로 빼앗지 않는 **비선점형**이다. 다음 프로세스를 고르는 기준이 달라도, 새 프로세스가 준비 상태에 들어왔다는 이유로 현재 실행을 중단시키지는 않는다.

NoteBase

보기별 해설

비선점형은 실행 중인 프로세스가 CPU 사용을 마치거나 입출력 대기 등으로 스스로 CPU를 반납할 때까지 실행을 보장한다. 선점형은 실행 중인 프로세스의 CPU를 운영체제가 회수하여 다른 프로세스에 배정할 수 있다.

ㄱ. FCFS — 비선점형: 준비 큐에 **먼저 도착한 순서**로 CPU를 배정한다. 일단 실행을 시작하면 뒤에 도착한 프로세스가 현재 프로세스의 CPU를 빼앗지 못한다.

ㄴ. HRN — 비선점형: 대기시간과 예상 실행시간을 반영한 응답률이 가장 높은 프로세스를 선택한다. 대기시간이 길어지면 선택될 우선순위가 높아지지만, 이것이 **실행 중인 프로세스를 선점한다는 뜻은 아니다**. 현재 프로세스가 CPU를 반납한 뒤 다음 실행 대상을 고른다.

ㄷ. RR — 선점형: 각 프로세스에 일정한 시간 할당량을 주고 번갈아 실행한다. **시간 할당량이 끝나면** 실행이 완료되지 않았어도 CPU를 회수하고 해당 프로세스를 준비 큐의 뒤로 보낸다.

ㄹ. SRT — 선점형: 남은 실행시간이 가장 짧은 프로세스를 실행한다. 현재 프로세스보다 **남은 실행시간이 짧은 프로세스가 준비되면** 현재 실행을 중단하고 CPU를 넘길 수 있다.

비선점형은 ㄱ과 ㄴ이므로, 이 둘을 묶은 ①에 해당한다.

NoteBase

문 13. 네트워크 접속 형태 중 트리형 토폴로지(topology)에 대한 설명으로 옳지 않은 것은?

① 네트워크의 확장이 용이하다.

② 병목 현상이 나타나지 않는다.

③ 분산처리 방식을 구현할 수 있다.

④ 중앙의 서버 컴퓨터에 장애가 발생하면 전체 네트워크에 영향을 준다.

정답 2번

간단한 해설

트리형에서는 여러 하위 노드의 통신이 상위 노드나 공통 연결 구간에 모이므로 **병목 현상이 발생할 수 있다**. 따라서 '병목 현상이 나타나지 않는다'는 설명은 잘못이다.

선지별 해설

트리형 토폴로지는 최상위 노드 아래에 하위 노드들이 가지처럼 연결되는 계층적 네트워크 구조이다. 서로 다른 가지 사이의 통신은 이들을 연결하는 상위 노드를 거친다.

① O: 기존 노드 아래에 새로운 하위 노드나 가지를 추가하여 **계층적으로 확장**할 수 있다. 네트워크 전체의 연결 구조를 다시 구성하지 않고도 규모를 늘리기 쉽다.

② X: 여러 하위 노드가 동시에 통신하면 상위 노드나 공통 연결 구간에 트래픽이 집중될 수 있다. 해당 구간의 처리 능력이나 전송 용량이 부족하면 데이터가 대기하는 **병목 현상이 발생한다**. 따라서 '병목 현상이 나타나지 않는다'는 '병목 현상이 발생할 수 있다'로 고쳐야 한다.

③ O: 각 하위 컴퓨터가 작업을 나누어 처리하고 네트워크를 통해 처리 결과를 주고받도록 구성할 수 있다. 이처럼 **여러 컴퓨터에 작업을 분담**시키는 분산처리 방식을 구현할 수 있다.

④ O: 최상위 중앙 서버를 중심으로 구성한 트리형에서는 하위 컴퓨터들이 그 서버의 기능이나 서비스에 의존한다. 따라서 중앙 서버에 장애가 발생하면 그 서비스를 이용하는 하위 컴퓨터들까지 영향을 받아 **전체 네트워크의 서비스 이용에 지장**을 줄 수 있다. 이는 모든 하위 컴퓨터 자체가 고장 난다는 뜻은 아니다.

NoteBase

문 14. IPv4 주소를 클래스별로 분류했을 때, B 클래스에 해당하는 것은?

① 12.23.34.45

② 111.111.11.11

③ 128.128.128.128

④ 222.111.222.111

정답 3번

간단한 해설

B 클래스는 IPv4 주소의 **첫 번째 옥텟이 128~191**인 범위이다. 128.128.128.128은 첫 번째 옥텟이 128이므로 이 범위에 속한다.

핵심 풀이

IPv4 주소에서 점으로 구분된 네 부분은 각각 8비트인 옥텟이다. 클래스는 **첫 번째 옥텟**으로 구분하며, B 클래스의 첫 번째 옥텟 범위는 128~191이다.

128.128.128.128의 첫 번째 옥텟은 128로, B 클래스 범위의 시작값이다. 뒤의 세 옥텟도 128이라는 사실은 클래스 판정에 영향을 주지 않는다.

다른 주소의 첫 번째 옥텟인 12와 111은 A 클래스 범위인 1~126에, 222는 C 클래스 범위인 192~223에 속한다. 따라서 B 클래스와 구별된다.

NoteBase

문 15. 다음 설명에 해당하는 모듈의 결합도는?

한 모듈이 다른 모듈의 내부 기능 및 자료를 직접 참조하거나 사용하는 경우로, 한 모듈에서 다른 모듈의 내부로 제어가 이동하는 경우도 이에 해당한다.

① 공통 결합도(common coupling)
② 내용 결합도(content coupling)
③ 외부 결합도(external coupling)
④ 자료 결합도(data coupling)

정답 2번

간단한 해설

다른 모듈의 **내부 기능이나 자료를 직접 참조·사용**하는 관계가 내용 결합도이다. 다른 모듈 내부의 명령으로 제어가 이동하는 경우도 여기에 해당한다.

공통 해설

내용 결합도(content coupling)는 한 모듈이 다른 모듈의 내부 기능이나 자료를 직접 참조·사용하여 **상대 모듈의 내부 구현에 의존**하는 관계이다. 다른 모듈의 내부 명령으로 제어를 직접 이동시키는 경우도 포함한다.

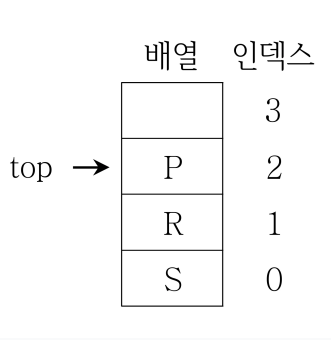
예를 들어 모듈 A가 모듈 B의 정해진 시작점에서 실행을 요청하는 대신 B의 중간 명령으로 직접 뛰어들면, A는 B의 내부 처리 구조에 의존하게 된다. 지문의 두 설명은 모두 이러한 내부 구현 의존을 나타낸다. 따라서 모듈 사이에 자료가 오간다는 사실보다 **다른 모듈의 내부에 직접 접근**한다는 점이 내용 결합도를 식별하는 기준이다.

NoteBase

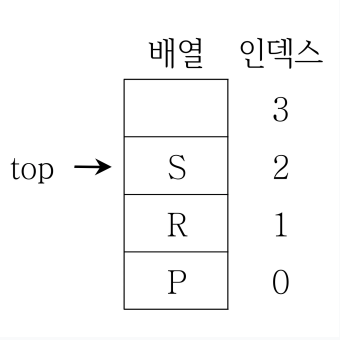
문 16. 다음 <정보>를 이용하여 아래에 주어진 <연산>을 차례대로 수행한 후의 스택 상태는?

<정보>
○ Create(s, n): 스택을 위한 크기 n의 비어 있는 배열 s를 생성하고, top의 값을 −1로 지정한다.
○ Push(s, e): top을 1 증가시킨 후, s[top]에 요소 e를 할당한다.
○ Pop(s): s[top]의 요소를 삭제한 후, top을 1 감소시킨다.
<연산>
Create(s, 4);
Push(s, 'S');
Push(s, 'T');
Pop(s);
Push(s, 'R');
Push(s, 'P');
Push(s, 'Q');
Pop(s);

①

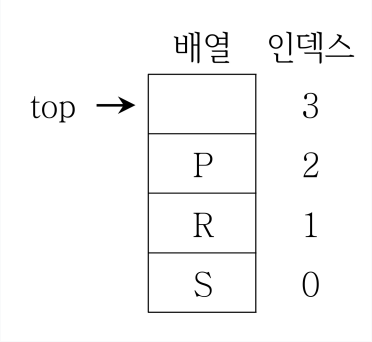


②

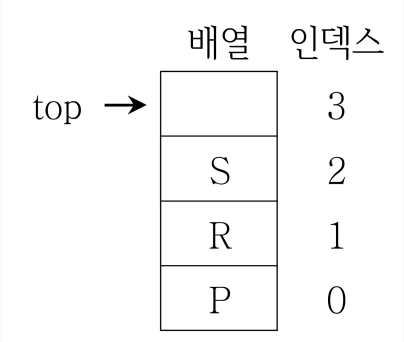


NoteBase

③



④



정답 1번

간단한 해설

Pop은 **가장 나중에 넣은 요소**를 삭제하므로, 첫 Pop에서는 T가, 마지막 Pop에서는 Q가 삭제된다. 최종적으로 인덱스 0·1·2에 S·R·P가 남고, **top은 2**가 된다.

NoteBase

계산 과정

top은 스택의 맨 위 요소가 있는 인덱스이다. 크기가 4인 배열의 인덱스는 0~3이며, Push는 top을 1 증가시킨 위치에 요소를 넣고 Pop은 현재 top의 요소를 삭제한 뒤 top을 1 감소시킨다. 따라서 **삽입·삭제 때마다 top과 해당 위치의 요소를 함께 추적**한다.

Create(s, 4): 빈 배열을 생성하고 top = −1로 둔다. 아직 들어 있는 요소가 없다는 뜻이다.

Push(s, 'S'): top은 −1 + 1 = 0이 되고, s[0]에 S가 들어간다.

Push(s, 'T'): top은 0 + 1 = 1이 되고, s[1]에 T가 들어간다. 아래에서 위로 S, T가 쌓인다.

첫 Pop(s): s[1]의 T를 삭제하고 top은 1 − 1 = 0이 된다. S는 s[0]에 그대로 남는다.

Push(s, 'R'): top은 0 + 1 = 1이 되고, **T가 삭제된 자리**인 s[1]에 R이 들어간다.

Push(s, 'P'): top은 1 + 1 = 2가 되고, s[2]에 P가 들어간다. 아래에서 위로 S, R, P가 쌓인다.

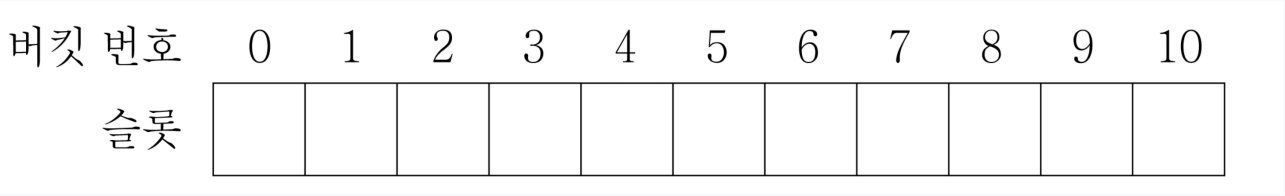
Push(s, 'Q'): top은 2 + 1 = 3이 되고, s[3]에 Q가 들어간다. 이때 배열의 네 칸이 모두 찬다.

마지막 Pop(s): 맨 위 s[3]의 Q를 삭제하고 top은 3 − 1 = 2가 된다. **다른 요소의 위치는 바뀌지 않는다**.

최종 상태는 s[0] = S, s[1] = R, s[2] = P이며, 인덱스 3의 칸은 비어 있다. top이 인덱스 2의 P를 가리키는 ①의 그림과 일치한다.

NoteBase

문 17. 다음은 전체 버킷 개수가 11개이고 버킷당 1개의 슬롯을 가지는 빈 해시 테이블이다. 입력키 12, 33, 13, 55, 23, 83, 11을 순서대로 저장하였을 때, 입력키 23이 저장된 버킷 번호는? (단, 해시 함수는 h(k)=k 11이고, 충돌 해결은 선형 조사법을 사용한다)



① 1

② 2

③ 3

④ 4

정답 4번

간단한 해설

23의 해시값은 1이지만, 먼저 저장된 키들이 버킷 1·2·3을 차지하고 있다. **선형 조사법은 차례로 다음 버킷을 확인**하므로 23은 처음으로 빈 버킷인 4에 저장된다.

NoteBase

계산 과정

그림의 버킷 번호는 0~10이며, 처음에는 모두 비어 있다. 각 버킷에는 키를 하나만 저장할 수 있다. 해시 함수 h(k)=k 11은 입력키 k를 11로 나눈 나머지를 최초 저장 후보 버킷 번호로 정한다. 선형 조사법은 충돌 시 **다음 번호의 버킷을 하나씩 확인하여 처음 만나는 빈 버킷에 저장**한다.

12 삽입: $h(12) = 1$이므로 빈 버킷 1에 저장한다.

33 삽입: $h(33) = 0$이므로 빈 버킷 0에 저장한다.

13 삽입: $h(13) = 2$이므로 빈 버킷 2에 저장한다.

55 삽입: $h(55) = 0$이지만 버킷 0에는 33이 있다. 다음 버킷 1과 2에도 각각 12와 13이 있으므로, 처음으로 빈 버킷인 3에 저장한다.

23 삽입: $h(23) = 1$이지만 버킷 1·2·3에는 각각 12·13·55가 있다. 따라서 1 → 2 → 3 → 4 순서로 조사하여 **빈 버킷 4**에 저장한다.

83과 11 삽입: $h(83) = 6$이므로 83은 빈 버킷 6에 저장한다. $h(11) = 0$이고 버킷 0~4가 모두 차 있으므로 11은 버킷 5에 저장한다. 이 과정에서 이미 저장된 23의 위치는 바뀌지 않는다.

따라서 입력키 23이 저장된 버킷 번호는 4이며, ④의 값과 일치한다.

NoteBase

문 18. 다음 파이썬 코드는 std 변수에 저장된 각각의 Student 객체에 대해 학생 id 및 국어, 영어 성적의 평균을 출력한다. (가) ~ (다)에 들어갈 내용을 바르게 연결한 것은?

```
class Student:
    def __init__(self, id, kor, eng):
        self.id = id
        self.kor = kor
        self.eng = eng

    def sum(self):
        return self.kor + self.eng

    def avg(self):
        return (가)

std = [
    Student("ok", 90, 100),
    Student("pk", 80, 90),
    Student("rk", 80, 80)
]

for to in (나) :
    print( (다) )
```

① (가): self.sum() / 2, (나): std, (다): to.id, to.avg()

② (가): self.sum() / 2, (나): Student, (다): Student.id, Student.avg()

③ (가): sum(self) / 2, (나): std, (다): to.id, to.avg(self)

④ (가): sum(self) / 2, (나): Student, (다): Student.id, Student.avg(self)

정답 1번

간단한 해설

객체를 통해 인스턴스 메서드를 호출하면 **그 객체가 self에 자동으로 전달**된다. 학생 객체가 담긴 std를 순회하며, 현재 객체인 to의 id와 평균 메서드에 접근해야 한다.

NoteBase

빈칸별 해설

(가): self.sum() / 2이다. avg의 self는 평균을 구할 학생 객체를 가리킨다. self.sum()은 **같은 객체의 sum 메서드**를 호출하여 self.kor + self.eng를 반환하므로, 이를 과목 수인 2로 나누면 평균이 된다. sum(self)는 이 메서드 호출이 아니라 내장 함수 sum에 객체를 전달하는 표현이다. 주어진 Student 객체는 순회 가능한 객체로 정의되어 있지 않으므로 이 방식으로 성적 합계를 구할 수 없다.

(나): std이다. std는 Student 객체 세 개를 담은 리스트이다. for to in std는 **리스트의 각 학생 객체**를 차례로 to에 대입한다. Student는 객체를 만드는 클래스이며, 이 코드에서 학생 객체 목록을 순회할 대상으로 사용할 수 없다.

(다): to.id, to.avg()이다. id는 생성자에서 self.id로 저장한 인스턴스 속성이므로 현재 학생 객체의 to.id로 읽는다. to.avg()를 호출하면 **to가 self에 자동으로 전달**되므로 괄호 안에 self를 다시 쓰지 않는다. to.avg(self)는 불필요한 인수를 하나 더 전달하는 형태이다. 또한 Student.id는 개별 학생의 속성을 읽는 방식이 아니며, Student.avg()는 메서드에 필요한 학생 객체를 전달하지 않은 호출이다.

(가)에 self.sum() / 2, (나)에 std, (다)에 to.id, to.avg()를 넣는 조합은 ①과 일치한다.

NoteBase

문 19. DBMS에서의 병행 수행 및 병행 제어에 대한 설명으로 옳은 것은?

① 2단계 로킹 규약을 적용하면 트랜잭션 스케줄의 직렬 가능성을 보장할 수 있으나 교착상태가 발생할 수도 있다.

② 트랜잭션이 데이터에 공용 lock 연산을 수행하면 해당 데이터에 read, write 연산을 모두 수행할 수 있다.

③ 연쇄 복귀는 하나의 트랜잭션이 여러 개의 데이터 변경 연산을 수행할 때 일관성 없는 상태의 데이터베이스에서 데이터를 가져와 연산을 수행함으로써 모순된 결과가 발생하는 것이다.

④ 갱신 분실은 트랜잭션이 완료되기 전에 장애가 발생하여 rollback 연산을 수행하면, 이 트랜잭션이 장애 발생 전에 변경한 데이터를 가져가 변경 연산을 수행한 또 다른 트랜잭션에도 rollback 연산을 수행하여야 한다는 것이다.

정답 1번

간단한 해설

2단계 로킹은 잠금을 획득하는 단계와 해제하는 단계를 구분하여 **충돌 직렬 가능성**을 보장한다. 다만 서로 상대가 보유한 잠금을 기다릴 수 있으므로 **교착상태는 발생할 수 있다**.

선지별 해설

① O: 2단계 로킹 규약은 잠금을 획득하되 해제하지 않는 확장 단계와, 잠금을 해제하되 새로 획득하지 않는 축소 단계로 나뉜다. 이 규약을 따르면 병행 실행의 결과가 어떤 직렬 실행과 같아지는 충돌 직렬 가능성을 보장한다. 그러나 **교착상태까지 방지하지는 못한다**. 예를 들어 두 트랜잭션이 각각 A와 B에 배타 잠금을 보유한 채 상대의 데이터에도 잠금을 요청하면 서로의 해제를 기다리게 된다.

② X: 공용 잠금은 데이터를 **읽기만 허용**하는 잠금이다. 여러 트랜잭션이 같은 데이터에 공용 잠금을 동시에 보유할 수 있지만, 데이터를 변경하려면 배타 잠금이 필요하다. 공용 잠금을 보유한 상태에서 쓰기를 하려면 배타 잠금으로 전환해야 한다.

③ X: 변경 도중의 일관성 없는 데이터를 읽어 모순된 결과를 얻는 것은 **불일치 분석**에 관한 설명이다. 연쇄 복귀는 아직 확정되지 않은 변경값을 다른 트랜잭션이 읽은 뒤, 그 값을 변경한 트랜잭션이 취소되면서 이를 읽은 트랜잭션도 함께 취소되어야 하는 현상이다. 예를 들어 두 데이터의 값을 함께 변경하는 도중 다른 트랜잭션이 변경된 값과 변경 전 값을 섞어 읽으면 일관성 없는 분석 결과를 얻을 수 있다.

④ X: 선지는 한 트랜잭션의 취소가 그 변경값을 읽은 다른 트랜잭션의 취소로 이어지는 연쇄 복귀를 설명한다. 갱신 분실은 여러 트랜잭션이 같은 데이터를 읽고 각각 변경하는 과정에서 **나중에 기록한 값이 앞선 갱신을 덮어써** 그 갱신 결과가 사라지는 현상이다. 따라서 갱신 분실의 핵심은 연쇄적인 rollback이 아니라 다른 트랜잭션의 갱신 결과가 보존되지 않는 데 있다.

NoteBase

문 20. 다음 C 프로그램의 출력 결과는?

```c
#include <stdio.h>

int recursive(int n) {
    int sum;
    if (n > 2) {
        sum = recursive(n-1) + recursive(n-2);
        printf("%d ", sum);
    }
    else
        sum = n;
    return sum;
}

int main(void) {
    int result;
    result = recursive(5);
    printf("%d", result);
    return 0;
}
```

① 1 2 3 5 7

② 1 3 5 7 9

③ 3 3 5 9 9

④ 3 5 3 8 8

정답 4번

간단한 해설

왼쪽 재귀 호출을 먼저 평가하면, 각 호출은 **하위 호출이 끝난 뒤 합을 출력**한다. recursive(5)가 8을 출력하고 반환한 뒤 main이 그 8을 한 번 더 출력한다.

NoteBase

계산 과정

평가 순서: ④의 출력은 덧셈식에서 **왼쪽의 recursive(n-1)을 먼저 평가하는 경우**에 해당한다. C 표준은 + 양쪽 피연산자의 평가 순서를 보장하지 않으므로, 아래에서는 이 조건으로 실행을 추적한다.

종료 조건과 출력 위치: n이 2 이하이면 sum에 n을 저장하여 반환하며 **출력은 하지 않는다**. 따라서 recursive(1)은 1, recursive(2)는 2를 반환한다. n이 2보다 크면 두 하위 호출의 반환값을 더한 다음 printf로 sum을 출력하고, 같은 값을 반환한다.

recursive(4)의 계산: main이 recursive(5)를 호출하면 먼저 recursive(4)를 계산한다. 그 안의 recursive(3)은 recursive(2)와 recursive(1)의 반환값을 더해 2 + 1 = 3을 출력하고 반환한다. 이어 recursive(2)가 2를 반환하므로 recursive(4)는 3 + 2 = 5를 출력하고 반환한다. 이때까지 출력된 값은 3 5이다.

recursive(5)의 나머지 계산: recursive(4)가 끝나면 오른쪽의 recursive(3)을 **새로 호출**한다. 이 호출도 2 + 1 = 3을 출력하고 반환한다. 이제 recursive(5)는 두 반환값을 더해 5 + 3 = 8을 출력하고 반환한다. 함수 내부에서 출력된 값은 3 5 3 8이다.

main의 출력: 반환된 8이 result에 저장된다. main의 printf가 result를 **한 번 더 출력**하므로 전체 출력은 3 5 3 8 8이며, ④와 일치한다.

NoteBase