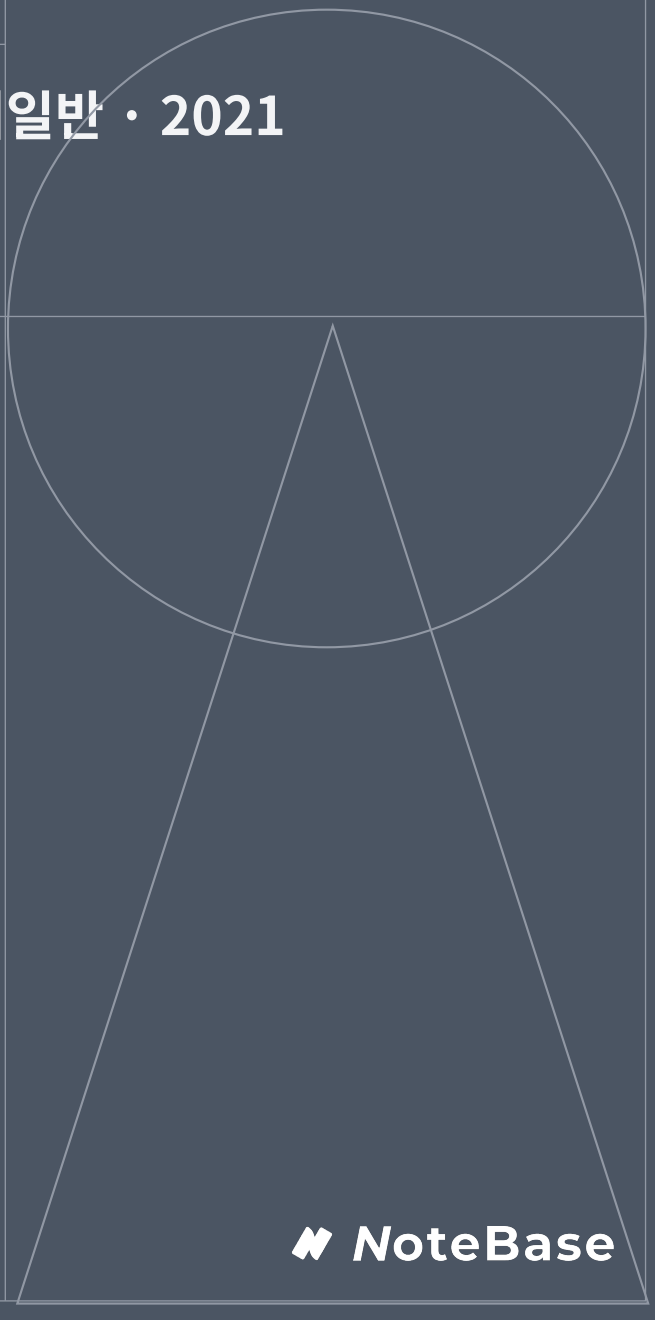
기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2021



NoteBase

기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2021

공무원/국가직/9급 · 컴퓨터일반 · 2021

문 1. 컴퓨팅 사고(Computational Thinking)에서 주어진 문제의 중요한 특징만으로 문제를 간결하게 재정의함으로써 문제 해결을 쉽게 하는 과정은?

① 분해

② 알고리즘

③ 추상화

④ 패턴 인식

정답 3번

간단한 해설

문제 해결에 필요한 **중요한 특징만 남겨** 문제를 간결하게 표현하는 과정이 추상화이다. 관련 없는 세부 사항을 제외하여 핵심에 집중할 수 있게 한다.

공통 해설

추상화는 복잡한 문제에서 **해결 목적에 필요한 특징은 남기고 불필요한 세부 사항은 제외**하여 문제를 단순하게 표현하는 과정이다. 중요한 정보까지 무조건 줄이는 것이 아니라, 무엇을 해결하려는지에 따라 남길 정보를 정한다. 문항의 '중요한 특징만으로 문제를 간결하게 재정의'한다는 설명은 이러한 추상화의 의미에 해당한다.

예를 들어 지하철로 목적지까지 가장 빨리 가는 경로를 찾으려면 역 사이의 연결 관계와 이동 시간 등의 정보가 필요하다. 반면 역 건물의 외벽 색깔은 이 문제를 푸는 데 필요하지 않다. 필요한 정보만 남겨 경로 문제로 표현하면, 현실의 복잡한 모습을 모두 다루지 않고도 해결에 집중할 수 있다.

NoteBase

문 2. 소프트웨어에 대한 설명으로 옳지 않은 것은?

① 하드웨어에 대응하는 개념으로 우리가 원하는 대로 컴퓨터를 작동하게 만드는 논리적인 바탕을 제공한다.

② 운영체제 등 컴퓨터 시스템을 가동시키는 데 사용되는 소프트웨어를 시스템 소프트웨어라 한다.

③ 문서 작성이나 게임 등 특정 분야의 업무를 처리하는 데 사용되는 소프트웨어를 응용 소프트웨어라 한다.

④ 고급 언어로 작성된 프로그램을 한꺼번에 번역한 후 실행하는 것이 인터프리터 방식이다.

정답 4번

간단한 해설

고급 언어 프로그램 전체를 한꺼번에 번역한 뒤 실행하는 것은 **컴파일러 방식**이다. 인터프리터 방식은 명령문을 해석하면서 실행한다.

선지별 해설

① O: 소프트웨어는 컴퓨터가 수행할 작업과 처리 절차를 지시하는 프로그램 등으로, 하드웨어를 원하는 목적에 맞게 작동시키는 **논리적 기반**이다. 하드웨어가 물리적인 장치라면 소프트웨어는 그 장치가 무엇을 어떻게 처리할지 정한다.

② O: 시스템 소프트웨어는 **컴퓨터의 자원을 관리하고 프로그램 실행을 지원**한다. 운영체제는 프로세서·메모리·입출력 장치 등을 관리하고 응용 프로그램이 실행될 환경을 제공하므로 시스템 소프트웨어에 해당한다.

③ O: 응용 소프트웨어는 **사용자의 특정 작업을 수행**하기 위한 프로그램이다. 문서 작성 프로그램은 문서 편집을, 게임 프로그램은 게임 실행을 목적으로 하므로 응용 소프트웨어에 해당한다.

④ X: 고급 언어로 작성된 프로그램 전체를 한꺼번에 번역한 뒤 실행하는 것은 **컴파일러 방식**이다. 인터프리터 방식은 프로그램 전체의 번역을 먼저 끝내는 대신 명령문을 해석하면서 실행한다. 선지의 '인터프리터 방식'을 '컴파일러 방식'으로 고쳐야 한다.

NoteBase

문 3. 4 GHz의 클록 속도를 갖는 CPU에서 CPI(Cycle per Instruction)가 4.0이고 총 10^{10}개의 명령어로 구성된 프로그램을 수행하려고 할 때, 이 프로그램의 실행 완료를 위해 필요한 시간은?

① 1초

② 10초

③ 100초

④ 1,000초

정답 2번

간단한 해설

명령어 수에 CPI를 곱한 **총 클록 사이클 수**를 클록 속도로 나누면 프로그램의 실행 시간이 된다. 주어진 값을 적용하면 **10초**가 걸린다.

계산 과정

CPI 4.0은 명령어 한 개를 수행하는 데 평균 4회의 클록 사이클이 필요하다는 뜻이다. 명령어가 10^{10}개이므로, 필요한 총 클록 사이클 수는 $10^{10} \times 4 = 4 \times 10^{10}$회이다.

클록 속도는 **1초 동안 발생하는 클록 사이클 수**이다. 1 GHz는 초당 10^{9}회이므로, 4 GHz는 초당 4×10^{9}회의 클록 사이클에 해당한다.

실행 시간은 **총 클록 사이클 수를 초당 클록 사이클 수로 나누어** 구한다. 따라서 실행 시간은 $\frac{4 \times 10^{10}}{4 \times 10^{9}} = 10$초이며, 이는 ②의 10초에 해당한다.

NoteBase

문 4. -35를 2의 보수(2’s Complement)로 변환하면?

① 11011100

② 11011101

③ 11101100

④ 11101101

정답 2번

간단한 해설

8비트에서 음수의 2의 보수 표현은 양수의 비트를 **모두 반전한 뒤 1을 더해** 구한다. +35의 00100011을 반전하면 11011100이고, 여기에 1을 더하면 -35의 표현인 11011101이 된다.

계산 과정

선택지가 모두 8자리이므로 **8비트**를 기준으로 -35를 표현한다. 2의 보수 방식에서 음수는 같은 비트 수로 나타낸 양수의 모든 비트를 반전한 뒤 1을 더해 구한다.

35를 이진수로 변환한다. 35 = 32 + 2 + 1이므로 이진수는 100011이다. 앞에 0을 채워 8비트로 나타내면 +35는 00100011이다.

00100011의 모든 0과 1을 서로 바꾸면 11011100이 된다. 이는 +35의 **1의 보수**이며, 아직 2의 보수를 구하는 과정에서 마지막 1을 더하지 않은 값이다.

11011100 + 1 = 11011101이다. 따라서 -35의 8비트 2의 보수 표현은 11011101이며, ②의 값과 일치한다.

NoteBase

문 5. OSI 7계층에서 계층별로 사용하는 프로토콜의 데이터 단위는 다음 표와 같다. ㉠~㉢에 들어갈 내용을 바르게 연결한 것은?

계층	데이터 단위
트랜스포트(Transport) 계층	(㉠)
네트워크(Network) 계층	(㉡)
데이터링크(Datalink) 계층	(㉢)
물리(Physical) 계층	비트

① ㉠: 세그먼트, ㉡: 프레임, ㉢: 패킷

② ㉠: 패킷, ㉡: 세그먼트, ㉢: 프레임

③ ㉠: 세그먼트, ㉡: 패킷, ㉢: 프레임

④ ㉠: 패킷, ㉡: 프레임, ㉢: 세그먼트

정답 3번

간단한 해설

송신할 때 상위 계층의 데이터에 각 계층의 제어 정보를 덧붙이는 **캡슐화**가 이루어진다. 네트워크 계층의 패킷을 데이터링크 계층에서 감싸면 **프레임**이 된다.

빈칸별 해설

㉠ 세그먼트: 트랜스포트 계층은 **프로세스 간 데이터 전달**을 담당한다. 대표적으로 TCP는 상위 계층의 데이터에 포트 번호 등을 담은 TCP 헤더를 붙여 세그먼트를 만든다.

㉡ 패킷: 네트워크 계층은 **목적지까지의 경로 선택과 전달**을 담당한다. IP는 전달받은 데이터에 출발지·목적지 IP 주소 등을 담은 IP 헤더를 붙여 패킷을 만든다.

㉢ 프레임: 데이터링크 계층은 **같은 링크로 연결된 노드 사이의 전송**을 담당한다. 이더넷에서는 네트워크 계층의 패킷에 MAC 주소 등을 담은 헤더와 오류 검출용 트레일러를 붙여 프레임을 만든다. 표의 물리 계층은 이를 비트 단위로 전송한다.

㉠ 세그먼트, ㉡ 패킷, ㉢ 프레임의 조합이므로 ③과 일치한다.

NoteBase

문 6. 300개의 노드로 이진 트리를 생성하고자 할 때, 생성 가능한 이진 트리의 최대 높이와 최소 높이로 모두 옳은 것은? (단, 1개의 노드로 생성된 이진 트리의 높이는 1이다)

① 최대 높이: 299, 최소 높이: 8

② 최대 높이: 299, 최소 높이: 9

③ 최대 높이: 300, 최소 높이: 8

④ 최대 높이: 300, 최소 높이: 9

정답 4번

간단한 해설

높이를 **경로에 포함된 노드 수**로 세므로, 300개를 한 줄로 연결하면 최대 높이는 300이다. 높이 8인 이진 트리에는 최대 255개만 들어가므로, 300개를 담는 최소 높이는 **9**이다.

계산 과정

문제에서 노드 1개의 높이를 1로 정했으므로, 루트를 1층으로 하여 가장 깊은 노드가 있는 층까지 센다. 즉, 높이는 루트에서 가장 깊은 노드까지의 경로에 포함된 노드 수이다.

최대 높이: 각 노드가 자식을 하나씩만 갖도록 300개를 한 줄로 연결하면, 루트부터 마지막 노드까지 모든 노드를 지나간다. 이진 트리는 자식을 최대 2개까지 가질 수 있으므로 **자식이 하나인 연결도 허용된다**. 따라서 최대 높이는 노드 수와 같은 300이다.

최소 높이: 높이를 줄이려면 위층부터 노드를 최대한 채운다. 각 노드가 자식을 최대 2개 가지므로 층별 최대 노드 수는 1, 2, 4, 8, …로 두 배씩 늘어난다. 따라서 높이가 h인 이진 트리에 들어갈 수 있는 최대 노드 수는 $2^h - 1$개이다. 높이 8에서는 $2^8 - 1 = 255$개로 300개를 담을 수 없지만, 높이 9에서는 $2^9 - 1 = 511$개까지 담을 수 있다. 실제로 8층까지 255개를 채우고 나머지 300 − 255 = 45개를 9층에 배치할 수 있으므로, **최소 높이는 9**이다.

최대 높이 300, 최소 높이 9이므로 ④의 값과 일치한다.

NoteBase

문 7. 아래와 같은 순서대로 회의실 사용 요청이 있을 때, 다음 중 가장 많은 회의실 사용 시간을 확보할 수 있는 스케줄링 방법은? (단, 회의실은 하나이고, 사용 요청은 (시작 시각, 종료 시각)으로 구성된다. 회의실에 특정 회의가 할당되면 이 회의 시간과 겹치는 회의 요청에 대해서는 회의실 배정을 할 수 없다)

(11:50, 12:30), (9:00, 12:00), (13:00, 14:30),
(14:40, 15:00), (14:50, 16:00), (15:40, 16:20),
(16:10, 18:00)

① 시작 시각이 빠른 요청부터 회의실 사용이 가능하면 확정한다.

② 종료 시각이 빠른 요청부터 회의실 사용이 가능하면 확정한다.

③ 사용 요청 순서대로 회의실 사용이 가능하면 확정한다.

④ 회의 시간이 긴 요청부터 회의실 사용이 가능하면 확정한다.

정답 4번

간단한 해설

비교 기준은 회의 횟수가 아니라 **총 사용 시간**이다. 주어진 요청에서 긴 회의부터 확정하면 서로 겹치지 않는 180분·90분·70분·110분의 회의를 확보하여 **총 450분(7시간 30분)**을 사용할 수 있다.

NoteBase

선지별 해설

각 방법이 확정한 회의들의 사용 시간을 더하여 비교한다. 회의 사이에 비어 있는 시간은 사용 시간에 포함하지 않으며, 검토 중인 요청이 이미 확정된 회의 중 하나라도 겹치면 제외한다.

① 시작 시각이 빠른 순서: (9:00, 12:00)을 먼저 확정하므로 이와 겹치는 (11:50, 12:30)은 제외된다. 이어 (13:00, 14:30), (14:40, 15:00)을 확정하고, 앞서 확정한 회의와 겹치는 (14:50, 16:00)은 제외한다. 이후 (15:40, 16:20)을 확정하면 (16:10, 18:00)은 겹쳐서 제외된다. 총 사용 시간은 180 + 90 + 20 + 40 = **330분(5시간 30분)**이다.

② 종료 시각이 빠른 순서: 가장 먼저 끝나는 (9:00, 12:00)을 확정하고, 이와 겹치는 (11:50, 12:30)은 제외한다. 이후 (13:00, 14:30), (14:40, 15:00)을 확정한다. (14:50, 16:00)은 이미 확정한 회의와 겹치므로 제외하고, (15:40, 16:20)을 확정한다. 마지막 (16:10, 18:00)은 겹쳐서 제외된다. 확정되는 회의가 ①과 같으므로 총 사용 시간도 **330분(5시간 30분)**이다.

③ 사용 요청 순서: 지문에 나열된 첫 요청인 (11:50, 12:30)을 먼저 확정하므로, 다음 요청인 180분짜리 (9:00, 12:00)은 겹쳐서 제외된다. 이후 (13:00, 14:30), (14:40, 15:00)을 확정하고, (14:50, 16:00)은 겹쳐서 제외한다. 이어 (15:40, 16:20)을 확정하면 (16:10, 18:00)은 제외된다. 총 사용 시간은 40 + 90 + 20 + 40 = **190분(3시간 10분)**이다.

④ 회의 시간이 긴 순서: 180분인 (9:00, 12:00), 110분인 (16:10, 18:00), 90분인 (13:00, 14:30), 70분인 (14:50, 16:00)을 차례로 검토하면 모두 서로 겹치지 않아 확정된다. 40분인 (11:50, 12:30)은 오전 회의와, 40분인 (15:40, 16:20)은 확정된 오후 회의들과 겹친다. 20분인 (14:40, 15:00)도 (14:50, 16:00)과 겹쳐 제외된다. 확정된 회의들을 시간순으로 보면 (9:00, 12:00), (13:00, 14:30), (14:50, 16:00), (16:10, 18:00)이며, 총 사용 시간은 180 + 90 + 70 + 110 = **450분(7시간 30분)**이다.

총 사용 시간은 ① 330분, ② 330분, ③ 190분, ④ 450분이므로, 주어진 요청에서는 ④의 방법이 가장 많은 사용 시간을 확보한다.

NoteBase

문 8. 제품 테이블에 대하여 SQL 명령을 실행한 결과가 다음과 같을 때, ㉠과 ㉡에 들어갈 내용을 바르게 연결한 것은?

<제품 테이블>

제품ID	제품이름	단가	제조업체
P001	나사못	100	A
P010	망치	1,000	B
P011	드라이버	3,000	B
P020	망치	1,500	C
P021	장갑	800	C
P022	너트	200	C
P030	드라이버	4,000	D
P031	절연테이프	500	D

<SQL 명령>

```
SELECT 제조업체, MAX(단가) AS 최고단가
FROM 제품
GROUP BY ( ㉠ )
HAVING COUNT(*) > ( ㉡ ) ;
```

<실행 결과>

제조업체	최고단가
B	3,000
C	1,500
D	4,000

NoteBase

① ㉠: 제조업체, ㉡: 1

② ㉠: 제조업체, ㉡: 2

③ ㉠: 단가, ㉡: 1

④ ㉠: 단가, ㉡: 2

정답 1번

간단한 해설

제조업체별로 그룹화하면 제품 수가 A는 1개, B·D는 2개, C는 3개이다. B·C·D만 남기려면 **COUNT(*) > 1**로 제품이 2개 이상인 그룹을 선택해야 한다.

계산 과정

㉠ 그룹화 기준: GROUP BY는 지정한 열의 값이 같은 행들을 하나의 그룹으로 묶는다. 실행 결과가 제조업체마다 최고단가를 보여 주므로 ㉠에는 **제조업체**가 들어간다. 단가로 그룹화하면 같은 가격의 제품끼리 묶이므로 주어진 결과를 얻을 수 없다.

제조업체별 집계: COUNT(*)는 각 그룹의 행 수, 즉 제품 수를 세고, MAX(단가)는 각 그룹에서 가장 큰 단가를 구한다. A는 제품 1개로 최고단가가 100이다. B는 단가 1,000과 3,000인 제품 2개로 최고단가가 3,000이다. C는 단가 1,500, 800, 200인 제품 3개로 최고단가가 1,500이다. D는 단가 4,000과 500인 제품 2개로 최고단가가 4,000이다.

㉡ 그룹 선택 조건: HAVING은 **그룹별 집계 결과에 조건을 적용**한다. A만 제외하고 B·C·D를 남기려면 제품 수가 1보다 큰 그룹을 선택해야 하므로 COUNT(*) > 1을 사용한다. COUNT(*) > 2로 하면 제품이 3개인 C만 남아 실행 결과와 다르다.

따라서 ㉠은 제조업체, ㉡은 1이며, 이는 ①의 연결과 일치한다.

NoteBase

문 9. 스택의 입력으로 4개의 문자 D, C, B, A가 순서대로 들어올 때, 스택 연산 PUSH와 POP에 의해서 출력될 수 없는 결과는?

① ABCD

② BDCA

③ CDBA

④ DCBA

정답 2번

간단한 해설

첫 출력이 B라면 D와 C는 이미 스택에 들어가 있다. B를 꺼낸 뒤에는 **C를 꺼내야 그 아래의 D를 꺼낼 수 있으므로**, B 다음에 D가 나오는 BDCA는 출력할 수 없다.

공통 해설

스택의 PUSH는 맨 위에 자료를 넣고, POP은 맨 위의 자료를 꺼내는 연산이다. 따라서 아직 꺼내지 않은 자료 사이에서는 **나중에 넣은 자료가 먼저 나오는 후입선출** 규칙을 따른다. 이 문제에서는 D, C, B, A의 입력 순서를 유지하면서 PUSH와 POP을 번갈아 수행할 수 있다.

ABCD: D, C, B, A를 모두 PUSH하면 맨 위에는 A가 놓인다. 이후 네 번 POP하면 A, B, C, D가 차례로 나오므로 생성할 수 있다.

BDCA: 첫 문자 B를 출력하려면 D와 C를 꺼내지 않은 채 D, C, B를 차례로 PUSH한 뒤 B를 POP해야 한다. 이때 남은 스택은 아래부터 D, C이므로 **D를 꺼내기 전에 반드시 C를 꺼내야 한다**. 남은 입력 A를 PUSH하더라도 C가 D를 가리고 있는 관계는 바뀌지 않는다. 따라서 두 번째 출력으로 D가 나올 수 없다.

CDBA: D와 C를 PUSH한 뒤 두 번 POP하면 C, D가 나온다. 이어서 B를 PUSH하고 POP한 다음 A를 PUSH하고 POP하면 전체 출력은 C, D, B, A가 된다.

DCBA: 입력 문자를 하나씩 PUSH한 직후 바로 POP하면 스택에 문자가 쌓이지 않는다. 따라서 입력과 같은 D, C, B, A 순서로 출력할 수 있다.

따라서 주어진 입력 순서를 지키며 PUSH와 POP으로 만들 수 없는 출력은 BDCA이다.

NoteBase

문 10. 임계구역에 대한 설명으로 옳은 것은?

① 임계구역에 진입하고자 하는 프로세스가 무한대기에 빠지지 않도록 하는 조건을 진행의 융통성(Progress Flexibility)이라 한다.

② 자원을 공유하는 프로세스들 사이에서 공유자원에 대해 동시에 접근하여 변경할 수 있는 프로그램 코드 부분을 임계영역(Critical Section)이라 한다.

③ 한 프로세스가 다른 프로세스의 진행을 방해하지 않도록 하는 조건을 한정 대기(Bounded Waiting)라 한다.

④ 한 프로세스가 임계구역에 들어가면 다른 프로세스는 임계구역에 들어갈 수 없도록 하는 조건을 상호 배제(Mutual Exclusion)라 한다.

정답 4번

간단한 해설

상호 배제는 공유자원을 다루는 임계구역에 **한 번에 하나의 프로세스만** 들어가도록 하는 조건이다. 한 프로세스가 실행 중이면 다른 프로세스는 그 프로세스가 임계구역을 나올 때까지 진입할 수 없다.

선지별 해설

① X: 진입을 요청한 프로세스가 무한히 기다리지 않도록 하는 조건은 **한정 대기**이다. 한 프로세스가 진입을 요청한 뒤 그 프로세스보다 먼저 다른 프로세스가 임계구역에 들어갈 수 있는 횟수에 상한을 두어 기아를 방지한다. 진행의 융통성은 임계구역이 비어 있고 진입하려는 프로세스가 있을 때 진입할 프로세스의 결정을 무한히 미루지 않아야 한다는 조건이다.

② X: 임계영역은 공유 데이터나 자원에 접근하여 읽거나 변경하는 프로그램 코드 부분이다. 여러 프로세스가 같은 공유자원을 동시에 변경하면 실행 순서에 따라 결과가 달라지는 경쟁 상태가 생길 수 있으므로, **동시 접근을 제한하여 보호해야 하는 구역**이다. 선지의 '동시에 접근하여 변경할 수 있는'을 동시 변경을 허용한다는 뜻으로 설명한 부분이 잘못되었다.

③ X: 한 프로세스가 다른 프로세스의 진행을 부당하게 방해하지 않아야 한다는 것은 **진행의 융통성**에 관한 설명이다. 특히 임계구역 밖에서 실행 중이며 진입을 원하지 않는 프로세스가 다른 프로세스의 임계구역 진입을 막아서는 안 된다. 한정 대기는 진입을 요청한 특정 프로세스가 다른 프로세스에 계속 밀려 무한히 기다리는 일을 방지하는 조건이다.

④ O: 상호 배제는 어떤 프로세스가 임계구역을 실행하는 동안 다른 프로세스가 같은 공유자원을 다루는 임계구역에 **동시에 진입하지 못하게** 하는 조건이다. 예를 들어 A가 공유 변수의 값을 변경하는 동안 B는 자신의 임계구역 진입을 기다려야 하며, A가 나온 뒤에야 진입할 수 있다.

NoteBase

문 11. 통합 테스팅 방법에 대한 설명으로 옳지 않은 것은?

① 연쇄식(Threads) 통합은 초기에 시스템 골격을 파악하기 어렵다.

② 빅뱅(Big-bang) 통합은 모든 모듈을 동시에 통합하여 테스팅한다.

③ 상향식(Bottom-up) 통합은 가장 하부 모듈부터 통합하여 테스팅한다.

④ 하향식(Top-down) 통합은 프로그램 제어 구조에서 상위 모듈부터 통합하는 것을 말한다.

정답 1번

간단한 해설

연쇄식 통합은 하나의 주요 기능을 수행하는 모듈들을 실행 흐름에 따라 연결하므로 **초기에 시스템 골격을 파악할 수 있다**. '초기에 시스템 골격을 파악하기 어렵다'는 상향식 통합의 특징이다.

선지별 해설

① X: 연쇄식 통합은 **주요 기능의 처리 경로에 속하는 모듈들**을 묶어 시험하고, 다른 기능의 경로를 점진적으로 추가하는 방식이다. 초기에는 시스템 골격에 해당하는 주요 경로부터 통합하므로 모듈 간 연결과 기본 동작을 일찍 확인할 수 있다. 따라서 '초기에 시스템 골격을 파악하기 어렵다'는 '초기에 시스템 골격을 파악할 수 있다'로 고쳐야 한다.

② O: 빅뱅 통합은 개별 시험을 마친 **모든 모듈을 한꺼번에 결합**하여 모듈 간 상호작용을 시험하는 방식이다. 여러 모듈이 동시에 연결되므로 오류가 발생했을 때 원인이 되는 모듈이나 연결 지점을 찾아내기 어렵다.

③ O: 상향식 통합은 **최하위 모듈부터 상위 모듈 방향**으로 결합하며 시험한다. 하위 모듈을 호출하는 상위 모듈이 아직 없으면 그 역할을 대신하는 시험용 프로그램인 드라이버를 사용한다. 전체 제어 구조는 상위 모듈이 결합되어야 드러나므로 초기에는 시스템 골격을 파악하기 어렵다.

④ O: 하향식 통합은 **최상위 제어 모듈부터 하위 모듈 방향**으로 결합하며 시험한다. 아직 결합하지 않은 하위 모듈은 호출에 대한 간단한 응답을 제공하는 임시 모듈인 스텁으로 대신하고, 이를 실제 모듈로 차례로 교체한다. 따라서 상위의 제어 구조를 초기에 확인할 수 있다.

NoteBase

문 12. 다음 중 파이썬 프로그래밍 언어에 대한 설명으로 옳은 것만을 모두 고르면?

ㄱ. 변수 선언 시 변수명 앞에 데이터형을 지정해야 한다.
ㄴ. 플랫폼에 독립적인 대화식 언어이다.
ㄷ. 클래스를 정의하여 객체 인스턴스를 생성할 수 있다.

① ㄴ
② ㄱ, ㄷ
③ ㄴ, ㄷ
④ ㄱ, ㄴ, ㄷ

정답 3번

간단한 해설

파이썬은 **변수명 앞에 자료형을 선언할 필요가 없는** 동적 타입 언어이다. 이름에 연결된 객체의 자료형은 실행 중에 결정되므로 'x = 10'처럼 값을 대입해 사용할 수 있다.

보기별 해설

ㄱ. X: 파이썬에서는 변수명 앞에 int나 float 같은 **자료형을 미리 선언하지 않는다**. 자료형은 변수 이름이 가리키는 객체에 속한다. 예를 들어 x = 10으로 정수를 연결한 뒤 x = '가'처럼 문자열을 다시 대입할 수 있다.

ㄴ. O: 파이썬은 **플랫폼 독립성과 대화식 실행**을 지원한다. Windows, Linux 등 각 플랫폼에 맞는 실행 환경을 갖추면 운영체제 전용 기능에 의존하지 않는 같은 소스 코드를 실행할 수 있다. 또한 대화식 환경에서는 명령을 입력하고 그 실행 결과를 바로 확인할 수 있다.

ㄷ. O: 파이썬은 객체 지향 프로그래밍을 지원하므로 **클래스를 정의하고 인스턴스를 생성**할 수 있다. class로 객체의 속성과 동작을 정의한 뒤, 클래스 이름에 괄호를 붙여 호출하면 그 클래스를 바탕으로 한 객체를 만들 수 있다.

옳은 보기는 ㄴ, ㄷ이며, 이 조합에 해당하는 선택지는 ③이다.

NoteBase

문 13. 해쉬(Hash)에 대한 설명으로 옳지 않은 것은?

① 연결리스트는 체이닝(Chaining) 구현에 적합하다.

② 충돌이 전혀 없다면 해쉬 탐색의 시간 복잡도는 O(1)이다.

③ 최악의 경우에도 이진 탐색보다 빠른 성능을 보인다.

④ 해쉬 함수는 임의의 길이의 데이터를 입력받을 수 있다.

정답 3번

간단한 해설

해시 탐색은 충돌이 한곳에 몰리는 최악의 경우 탐색 시간이 $O(n)$까지 늘어날 수 있다. 따라서 최악의 경우에도 $O(\log n)$인 이진 탐색보다 **항상 빠르다고 할 수 없다**.

선지별 해설

① O: 체이닝은 서로 다른 키가 같은 해시 주소에 대응하는 충돌을 처리하는 방법이다. 같은 주소에 속하는 자료를 **연결리스트로 이어 저장**하면, 그 주소에서 연결된 노드를 따라가며 원하는 키를 찾을 수 있다.

② O: 해시값 계산과 한 번의 키 비교를 상수 시간으로 보는 일반적인 분석에서, **충돌이 없으면 계산한 주소로 바로 접근**하여 원하는 자료를 확인할 수 있다. 저장된 자료 수가 늘어도 확인할 후보 수가 늘지 않으므로 탐색 시간 복잡도는 $O(1)$이다.

③ X: 자료가 n개일 때, 체이닝에서 모든 키가 같은 주소로 몰리면 원하는 자료를 찾기 위해 연결리스트의 노드를 최대 n개 확인해야 하므로 최악의 탐색 시간은 $O(n)$이다. 정렬된 배열의 이진 탐색은 비교할 때마다 탐색 범위를 절반으로 줄여 최악에도 $O(\log n)$이므로, 해시 탐색이 **최악의 경우에도 더 빠르다는 보장은 없다**.

④ O: 해시 함수는 **길이가 서로 다른 입력**을 해시값으로 변환할 수 있다. 예를 들어 길이가 다른 문자열들을 해시 테이블의 정해진 주소 범위에 대응시킬 수 있으므로 입력 데이터의 길이가 일정할 필요는 없다.

NoteBase

문 14. 프로세스의 메모리는 세그먼테이션에 의해 그 역할이 할당되어 있다. 표준 C언어로 작성된 프로그램이 컴파일 후 실행파일로 변환되어 메모리를 할당받았을 때, 이 프로그램에 할당된 세그먼트에 대한 설명으로 옳은 것은?

① 데이터 세그먼트는 모든 서브루틴의 지역변수와 서브루틴 종료 후 돌아갈 명령어의 주소값을 저장한다.

② 스택은 현재 실행 중인 서브루틴의 매개변수와 프로그램의 전역변수를 저장한다.

③ 코드 세그먼트는 CPU가 실행할 명령어와 메인 서브루틴의 지역변수를 저장한다.

④ 힙(Heap)은 동적 메모리 할당을 위해 사용되는 공간이고, 주소값이 커지는 방향으로 증가한다.

정답 4번

간단한 해설

힙은 실행 중 malloc() 등으로 요청하는 **동적 메모리**를 할당하는 영역이다. 전형적인 프로세스 메모리 배치에서는 힙 영역이 확장될 때 **낮은 주소에서 높은 주소 방향**으로 커진다.

선지별 해설

이 문항은 프로세스 메모리를 코드·데이터·힙·스택으로 구분하는 전형적인 배치를 전제로 한다.

① X: 서브루틴의 일반적인 지역변수와 복귀 주소를 저장하는 곳은 **스택**이다. 복귀 주소는 서브루틴 실행을 마친 뒤 호출한 곳의 어느 명령어부터 실행을 이어 갈지 나타낸다. 데이터 영역은 전역변수와 정적변수를 저장하며, 초기화 여부 등에 따라 초기화된 데이터 영역과 BSS 영역으로 구분한다.

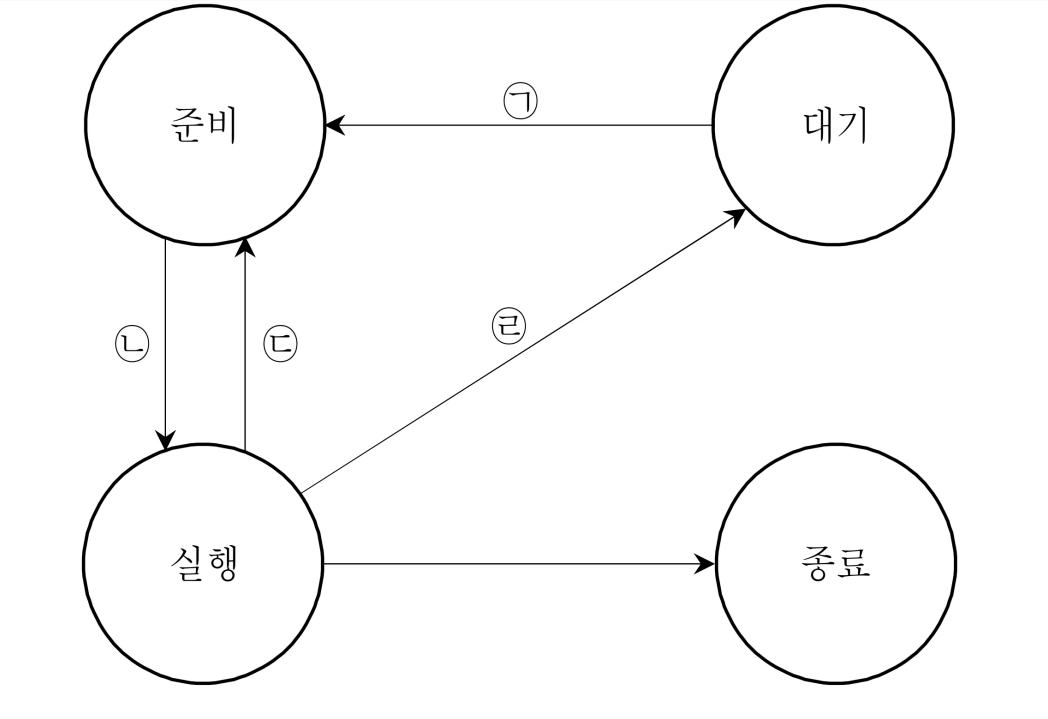
② X: 전형적인 함수 호출 구조에서 매개변수는 스택에 저장할 수 있지만, **전역변수는 데이터 영역**에 저장한다. 스택에는 함수 호출별로 필요한 매개변수, 자동 지역변수, 복귀 주소 등의 정보가 놓인다. 전역변수는 특정 함수 호출이 끝나더라도 유지되므로 함수 호출마다 생성·정리되는 스택의 지역변수와 구별된다.

③ X: 코드 세그먼트는 **CPU가 실행할 명령어**를 저장한다. main 함수도 다른 함수와 마찬가지로 자동 지역변수는 스택에 저장하므로, 메인 서브루틴의 지역변수를 코드 세그먼트에 저장한다는 부분이 잘못되었다. 함수 안에서 선언했더라도 static 지역변수는 자동 지역변수와 달리 데이터 영역에 저장한다.

④ O: 힙은 프로그램 실행 중 필요한 메모리를 동적으로 할당하는 공간이다. 예를 들어 malloc()으로 확보한 메모리는 힙에서 관리하며, 사용이 끝나면 free()로 해제한다. 이 문항이 전제하는 전형적인 메모리 배치에서는 힙이 **주소값이 커지는 방향**으로 확장된다. 이는 힙 영역의 확장 방향에 관한 설명이며, malloc()을 호출할 때마다 반환되는 주소가 반드시 커진다는 뜻은 아니다. 실제 메모리 배치와 확장 방식은 실행 환경에 따라 달라질 수 있다.

NoteBase

문 15. 다음은 프로세스 상태 전이도이다. 각 상태 전이에 대한 예로 적절하지 않은 것은?



① ㉠ - 프로세스에 자신이 기다리고 있던 이벤트가 발생하였다.

② ㉡ - 실행할 프로세스를 선택할 때가 되면, 운영체제는 프로세스들 중 하나를 선택한다.

③ ㉢ - 실행 중인 프로세스가 자신에게 할당된 처리기의 시간을 모두 사용하였다.

④ ㉣ - 실행 중인 프로세스가 작업을 완료하거나 실행이 중단되었다.

정답 4번

간단한 해설

㉣은 실행 중인 프로세스가 **입출력 완료 등 특정 이벤트를 기다리기 위해 대기 상태로 이동**하는 전이이다. 작업을 완료하거나 강제로 종료되면 대기 상태가 아니라 **종료 상태**로 이동한다.

NoteBase

선지별 해설

① O: ㉠은 대기 → 준비 전이이다. 입출력 완료처럼 기다리던 이벤트가 발생하면 대기 원인이 해소된다. 다만 처리기를 즉시 배정받는 것은 아니므로, **실행 가능한 상태에서 처리기 배정을 기다리는 준비 상태**로 이동한다.

② O: ㉡은 준비 → 실행 전이이다. 운영체제가 준비 상태의 프로세스 중 하나를 선택하여 **처리기를 배정**하면, 그 프로세스가 실제 명령을 수행하는 실행 상태가 된다.

③ O: ㉢은 실행 → 준비 전이이다. 할당된 처리기 시간을 모두 사용하면 처리기를 회수당하지만, 입출력 완료 같은 이벤트를 기다려야 하는 것은 아니다. 따라서 다시 처리기를 배정받으면 실행할 수 있는 **준비 상태**로 돌아간다.

④ X: ㉣은 실행 → 대기 전이이다. 실행 중 입출력을 요청하는 등 **특정 이벤트가 발생해야 실행을 계속할 수 있는 경우**에 이 전이가 일어난다. 작업을 완료하거나 실행이 강제로 끝난 경우에는 실행 → 종료 전이가 일어나므로, 선지의 설명은 그림 아래쪽의 종료 상태로 향하는 화살표에 해당한다.

NoteBase

문 16. -30.25×2^{-8}**의 값을 갖는 IEEE 754 단정도(Single Precision) 부동소수점(Floating-point) 수를 16진수로 변환하면?**

① 5DF30000

② 9ED40000

③ BDF20000

④ C8F40000

정답 3번

간단한 해설

정규화한 값은 $-1.111001_2 \times 2^{-4}$이며, 부호 비트는 1이고 저장 지수는 123이다. **선행 1을 제외한 23비트 분수부**를 이어 붙이면 16진수 BDF20000이 된다.

계산 과정

IEEE 754 단정도는 상위 비트부터 부호 1비트, 지수 8비트, 분수부 23비트로 구성된다. 정규수는 $\pm(1.f)_2 \times 2^e$ 형태로 나타내며, e는 실제 지수이고 f는 소수점 아래 비트열이다. 지수부에는 실제 지수에 편향값 127을 더해 저장한다.

1. 이진수 변환과 정규화: 30은 이진수 11110이고 0.25는 이진수 0.01이므로 $30.25_{10} = 11110.01_2$이다. 이를 정규화하면 $30.25_{10} = 1.111001_2 \times 2^4$이다. 여기에 주어진 2^{-8}을 곱하면 실제 지수는 $4 - 8 = -4$가 되어, 전체 값은 $-1.111001_2 \times 2^{-4}$이다.

2. 부호 비트: 주어진 값이 음수이므로 부호 비트는 1이다. 부호는 별도 비트로 나타내므로, 양수의 전체 비트열에 2의 보수를 취하지 않는다.

3. 지수부: **실제 지수에 127을 더한 값**을 저장하므로 $-4 + 127 = 123$이다. 따라서 8비트 지수부는 $123_{10} = 01111011_2$이다.

4. 분수부: 정규화된 유효숫자 1.111001에서 **소수점 앞의 1은 생략**하고 소수점 아래 111001만 저장한다. 뒤에 0을 17개 채워 23비트로 만들면 11100100000000000000000이다. 유효한 비트가 모두 들어가므로 반올림은 필요 없다.

5. 16진수 변환: 부호·지수·분수부를 순서대로 이어 붙이고 최상위 비트부터 4비트씩 묶으면 1011 1101 1111 0010 0000 0000 0000 0000이다. 각 묶음은 16진수로 B D F 2 0 0 0 0이므로, ③의 BDF20000과 일치한다.

NoteBase

문 17. 다음은 어느 학생이 C 언어로 작성한 학점 계산 프로그램이다. 출력 결과는?

```
#include <stdio.h>
int main()
{
  int score = 85;
  char grade;
  if (score >= 90) grade='A';
  if (score >= 80) grade='B';
  if (score >= 70) grade='C';
  if (score < 70)  grade='F';
  printf("학점 : %c\n", grade);
  return 0;
}
```

① 학점: A

② 학점: B

③ 학점: C

④ 학점: F

정답 3번

간단한 해설

각 if문은 **서로 독립적으로 실행**된다. score가 85이므로 grade에 'B'가 저장된 뒤, 70 이상 조건도 참이어서 **'C'로 덮어써진다**.

NoteBase

계산 과정

초기 상태: score는 85이며 프로그램 실행 중 바뀌지 않는다. grade는 학점을 저장할 문자형 변수이다. 네 조건문은 else if로 연결되어 있지 않으므로 **앞의 조건이 참이어도 뒤의 if문을 계속 검사**한다.

90 이상 조건: 85는 90 이상이 아니므로 grade에 'A'를 대입하는 문장은 실행되지 않는다.

80 이상 조건: 85는 80 이상이므로 grade에 'B'를 저장한다. 이 시점의 grade 값은 'B'이다.

70 이상 조건: 85는 70 이상이기도 하므로 grade에 'C'를 저장한다. 같은 변수에 다시 대입하므로 **기존 값 'B'가 'C'로 바뀐다**.

70 미만 조건: 85는 70 미만이 아니므로 'F'를 대입하지 않는다. 따라서 grade는 'C'를 유지한다.

출력: printf의 %c는 grade에 저장된 문자 하나를 출력한다. 최종 출력은 '학점 : C'이며, 이는 ③의 '학점: C'에 해당한다.

NoteBase

문 18. 파이프라인 해저드(Pipeline Hazard)에 대한 다음 설명에서 ㉠과 ㉡에 들어갈 내용을 바르게 연결한 것은?

○ 하드웨어 자원의 부족 때문에 명령어를 적절한 클록 사이클에 실행할 수 있도록 지원하지 못할 때 (㉠) 해저드가 발생한다.
○ 실행할 명령어를 적절한 클록 사이클에 가져오지 못할 때 (㉡) 해저드가 발생한다.

① ㉠: 구조적, ㉡: 제어
② ㉠: 구조적, ㉡: 데이터
③ ㉠: 데이터, ㉡: 구조적
④ ㉠: 데이터, ㉡: 제어

정답 1번

간단한 해설

하드웨어 자원의 부족·충돌로 명령어 실행이 지연되면 구조적 해저드이다. 분기 등으로 **다음에 가져올 명령어가 확정되지 않아** 인출이 지연되면 제어 해저드이다.

빈칸별 해설

파이프라인 해저드는 여러 명령어의 처리 단계를 겹쳐 실행하는 과정에서, 명령어가 예정된 클록 사이클에 진행하지 못하게 되는 상황이다.

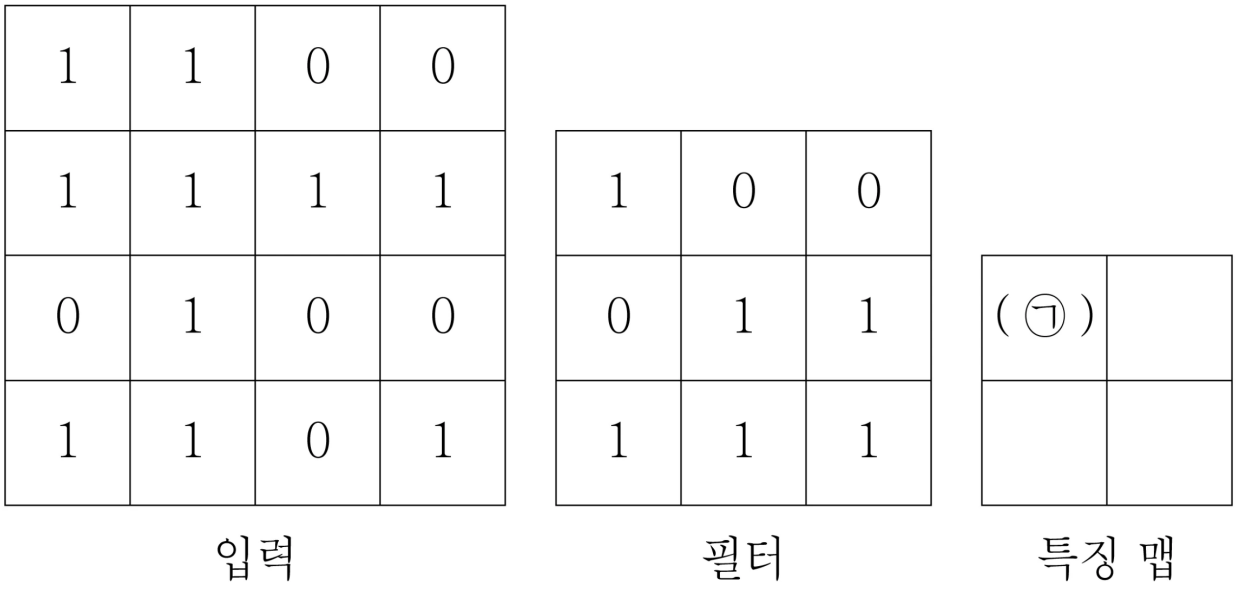
㉠ 구조적 해저드: 파이프라인의 여러 단계가 같은 클록 사이클에 동일한 하드웨어 자원을 요구하지만, 그 자원이 모든 요구를 동시에 처리하지 못할 때 발생한다. 예를 들어 명령어 인출과 데이터 접근이 하나의 메모리 포트를 함께 사용해야 한다면 한쪽은 기다려야 한다. 따라서 제시문의 **하드웨어 자원 부족**은 구조적 해저드에 해당한다.

㉡ 제어 해저드: 분기 명령어 등으로 실행 흐름이 바뀔 수 있는데, 분기 여부나 목적지가 아직 결정되지 않으면 다음에 실행할 명령어를 제때 가져오기 어렵다. 핵심은 **다음 명령어의 주소가 확정되지 않은 것**이다. 반면 데이터 해저드는 뒤 명령어가 필요로 하는 앞 명령어의 연산 결과가 아직 준비되지 않는 등 데이터 의존성 때문에 발생한다.

따라서 ㉠은 구조적, ㉡은 제어이며, 이 연결은 ①과 일치한다.

NoteBase

문 19. 합성곱 신경망(CNN, Convolutional Neural Network) 처리 시 다음과 같은 입력과 필터가 주어졌을 때, 합성곱에 의해 생성된 특징 맵(Feature Map)의 ㉠에 들어갈 값은?



① 3
② 4
③ 5
④ 6

정답 2번

간단한 해설

입력의 왼쪽 위 3×3 영역과 필터에서 **같은 위치의 값들을 곱해 더한다**. 각 행에서 얻은 합이 1, 2, 1이므로 ㉠의 값은 4이다.

NoteBase

계산 과정

CNN의 합성곱은 입력의 일부 영역에 필터를 겹쳐 놓고, 같은 위치의 값끼리 곱한 뒤 그 결과를 모두 더해 특징 맵의 한 칸을 구한다. ㉠은 특징 맵의 왼쪽 위 칸이므로 **입력의 1~3행 · 1~3열**에 3×3 필터를 표시된 방향 그대로 겹친다. 이 입력 영역의 각 행은 (1, 1, 0), (1, 1, 1), (0, 1, 0)이다.

필터의 각 행은 (1, 0, 0), (0, 1, 1), (1, 1, 1)이다. **필터가 0인 위치의 곱은 0**이므로, 필터가 1인 위치의 입력값만 합에 기여한다. 행별로 곱의 합을 구하면 다음과 같다.
첫째 행: 1×1 + 1×0 + 0×0 = 1
둘째 행: 1×0 + 1×1 + 1×1 = 2
셋째 행: 0×1 + 1×1 + 0×1 = 1

세 행의 결과를 모두 더하면 1 + 2 + 1 = 4이다. 따라서 ㉠에 들어갈 값은 4이며, 선택지 ②에 해당한다.

NoteBase

문 20. 해밍코드에 대한 패리티 비트 생성 규칙과 인코딩 예가 다음과 같다. 이에 대한 설명으로 옳은 것은?

<패리티 비트 생성 규칙>

원본 데이터	d4	d3	d2	d1			
인코딩된 데이터	d4	d3	d2	p4	d1	p2	p1

$$p1 = (d1 + d2 + d4) \bmod 2$$

$$p2 = (d1 + d3 + d4) \bmod 2$$

$$p4 = (d2 + d3 + d4) \bmod 2$$

<인코딩 예>

원본 데이터	0	0	1	1			
인코딩된 데이터	0	0	1	1	1	1	0

① 이 방법은 홀수 패리티를 사용하고 있다.

② 원본 데이터가 0100이면 0101110으로 인코딩된다.

③ 패리티 비트에 오류가 발생하면 복구는 불가능하다.

④ 수신측이 0010001을 수신하면 한 개의 비트 오류를 수정한 후 최종적으로 0010으로 복호한다.

정답 4번

간단한 해설

수신값 0010001은 짝수 패리티 검사에서 **4번 위치의 패리티 비트 오류**가 확인된다. 이 비트를 0에서 1로 바꾼 0011001에서 패리티 비트를 제외하면 원본 데이터 0010을 얻는다.

NoteBase

선지별 해설

인코딩된 비트의 배열은 왼쪽부터 $d_4, d_3, d_2, p_4, d_1, p_2, p_1$이다. 오른쪽 끝을 1번으로 세면 패리티 비트는 1·2·4번 위치, 데이터 비트는 7·6·5·3번 위치에 놓인다.

① X: 주어진 규칙은 **짝수 패리티**를 사용한다. 각 패리티 비트는 담당 데이터 비트의 합을 2로 나눈 나머지이므로, 데이터의 1 개수가 홀수이면 1을 붙이고 짝수이면 0을 붙인다. 따라서 패리티 비트까지 포함한 각 검사 묶음의 1 개수가 짝수가 된다. 제시된 인코딩 예 0011110에서도 각 검사 묶음의 1 개수는 2개이다.

② X: 원본 데이터 0100은 $d_4 = 0, d_3 = 1, d_2 = 0, d_1 = 0$이다. 생성 규칙에 따라 p_1=(0+0+0) 2=0, p_2=(0+1+0) 2=1, p_4=(0+1+0) 2=1이다. 이를 정해진 위치에 넣으면 **0101010**으로 인코딩된다. 선지의 0101110은 데이터 비트인 3번 위치까지 1로 바뀌어 있어 원본 데이터 0100의 인코딩 결과가 아니다.

③ X: 한 비트 오류라면 **패리티 비트도 수정할 수 있다**. 1번 패리티 비트는 1번 검사에만, 2번 패리티 비트는 2번 검사에만, 4번 패리티 비트는 4번 검사에만 포함된다. 따라서 패리티 비트 하나가 뒤집히면 해당 검사만 실패하므로 오류 위치를 찾아 그 비트를 반전할 수 있다.

④ O: 수신값 0010001의 1번 검사 대상은 1·3·5·7번 위치이며 1이 2개라서 통과한다. 2번 검사 대상은 2·3·6·7번 위치이며 1이 0개라서 통과한다. 4번 검사 대상은 4·5·6·7번 위치이며 1이 1개라서 실패한다. 한 비트 오류를 전제로 하면 **4번 검사에만 포함되는 4번 위치**의 비트가 오류이다. 그 비트를 0에서 1로 바꾸면 0011001이 된다. 여기서 1·2·4번의 패리티 비트를 제외하고 7·6·5·3번의 데이터 비트를 순서대로 읽으면 0010으로 복호된다.

NoteBase