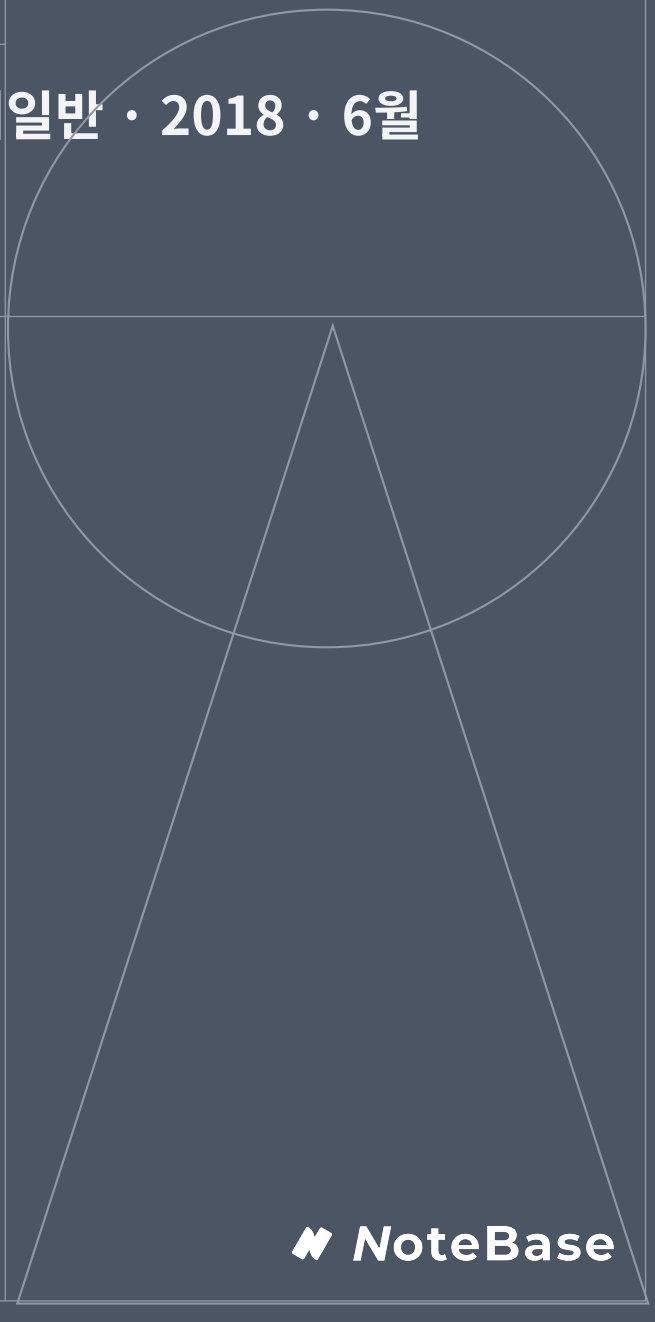
기출OX 해설집

공무원/서울시/9급 · 컴퓨터일반 · 2018 · 6월



NoteBase

기출OX 해설집

공무원/서울시/9급 · 컴퓨터일반 · 2018 · 6월

공무원/서울시/9급 · 컴퓨터일반 · 2018 · 6월

문 1. 중위 표기법으로 표현된 <보기>의 수식을 후위 표기법으로 옳게 표현한 것은?

<보기>

$a + (b * c - d) * (e - f * g) - h$

① ab * cd + efg * − * − h −

② abc * d + ef * g − * − h −

③ abcd * − efg * + * − h −

④ abc * d − efg * − * + h −

정답 4번

간단한 해설

후위 표기법은 **연산자를 두 피연산자 뒤에** 놓는다. 두 괄호의 결과를 곱한 뒤 a와 더하고 마지막에 h를 빼므로, 바깥쪽 덧셈의 + 뒤에 h −가 이어진다.

NoteBase

계산 과정

후위 표기법은 '왼쪽 피연산자, 오른쪽 피연산자, 연산자' 순서로 나타낸다. 부분식도 하나의 피연산자로 취급하며, **뺄셈의 좌우 피연산자 순서를 유지**해야 한다. 주어진 식은 괄호 안의 곱셈을 먼저 처리하고, 두 괄호의 결과를 곱한다. 바깥쪽 +와 −는 우선순위가 같으므로 왼쪽부터 적용한다.

첫째 괄호: $b \cdot c - d$에서 b와 c를 곱하는 부분은 b c *이다. 이 결과에서 d를 빼므로 후위 표기는 b c * d −이다.

둘째 괄호: $e - f \cdot g$에서 f와 g의 곱은 f g *이다. e에서 그 곱을 빼므로 후위 표기는 e f g * −이다. **e를 먼저 놓고 곱에 해당하는 부분을 뒤에** 놓아야 뺄셈의 방향이 보존된다.

두 괄호의 곱: 첫째 괄호의 후위 표기와 둘째 괄호의 후위 표기를 차례로 적고, 두 결과를 곱하는 *를 마지막에 붙인다. 따라서 b c * d − e f g * − *가 된다.

a와의 덧셈: a가 왼쪽 피연산자이고 두 괄호의 곱이 오른쪽 피연산자이다. a 다음에 곱의 후위 표기를 적고 +를 붙이면 a b c * d − e f g * − * +가 된다.

마지막 뺄셈: 앞서 구한 합에서 h를 빼므로 **합 전체의 후위 표기 뒤에 h −**를 붙인다. 완성된 후위 표기는 a b c * d − e f g * − * + h −이며, ④의 표현과 일치한다.

NoteBase

문 2. 소프트웨어 개발 프로세스 모델에 대한 설명으로 가장 옳지 않은 것은?

① 폭포수(Waterfall) 모델은 단계별 정형화된 접근 방법 및 체계적인 문서화가 용이하다.

② RAD(Rapid Application Development) 모델은 CASE(Computer Aided Software Engineering) 도구를 활용하여 빠른 개발을 지향한다.

③ 나선형(Spiral) 모델은 폭포수(Waterfall) 모델과 원형(Prototype) 모델의 장점을 결합한 모델이다.

④ 원형(Prototype) 모델은 고객의 요구를 완전히 이해하여 개발을 진행하는 것으로 시스템 이해도가 높은 관리자가 있는 경우 유용하다.

정답 4번

간단한 해설

원형 모델은 고객의 요구를 처음부터 완전히 이해한 상태를 전제로 하지 않는다. **요구사항이 불명확할 때** 시제품을 보여 주고 고객의 피드백을 받아 요구사항을 구체화하는 데 유용하다.

선지별 해설

① O: 폭포수 모델은 요구사항 분석, 설계, 구현, 시험 등의 단계를 순차적으로 진행한다. 단계마다 수행할 작업과 산출물이 정해져 있어 **단계별 관리와 체계적인 문서화**가 용이하다.

② O: RAD 모델은 개발 기간을 단축하여 빠르게 시스템을 구축하는 것을 지향한다. 분석·설계·코드 생성 등의 개발 작업을 지원하거나 자동화하는 **CASE 도구**를 활용하여 개발 속도를 높인다.

③ O: 나선형 모델은 폭포수 모델의 체계적인 개발 절차와 원형 모델의 시제품 제작·피드백 방식을 결합한다. 개발을 반복하면서 **위험을 분석하고 줄이는 것**이 특징이며, 필요한 경우 시제품을 만들어 기술적 가능성이나 요구사항을 확인한다.

④ X: 원형 모델은 고객의 요구를 완전히 이해한 뒤 개발하는 방식이 아니라, **시제품과 고객 피드백으로 요구사항을 구체화**하는 방식이다. 고객이 원하는 기능이나 화면을 명확히 설명하기 어려울 때 시제품을 사용해 보면서 누락되거나 잘못 이해된 요구를 발견할 수 있다. 따라서 요구사항이 불명확한 경우에 유용하며, 시스템 이해도가 높은 관리자의 존재를 적용 조건으로 삼는 것은 적절하지 않다.

NoteBase

문 3. 서로 다른 시스템 간의 통신을 위한 표준을 제공함으로써 통신에 방해가 되는 기술적인 문제점을 제거하고 상호 인터페이스를 정의한 OSI 참조 모델의 계층에 대한 설명으로 가장 옳지 않은 것은?

① 네트워크 계층은 물리 계층에서 전달받은 데이터에 대한 동기를 확인하는 기능, 데이터의 원활한 전송을 제어하는 흐름제어(Flow Control) 기능, 안전한 데이터 전송을 위한 에러 제어(Error Control) 기능을 수행한다.

② 물리 계층은 상위 계층으로부터 전달받은 데이터의 물리적인 링크를 설정하고 유지, 해제하는 기능을 담당한다.

③ 전송 계층은 통신하고 있는 두 사용자 사이에서 데이터 전송의 종단 간(end-to-end) 서비스 질을 높이고 신뢰성을 제어하는 기능을 담당한다.

④ 응용 계층은 사용자가 직접 접하는 부분이며 전자 메일 서비스, 파일 전송 서비스, 네트워크 관리 등이 있다.

정답 1번

간단한 해설

물리 계층에서 받은 비트열의 프레임 경계를 확인하고 흐름 제어와 오류 제어를 수행하는 것은 **데이터 링크 계층**이다. 네트워크 계층은 목적지까지 패킷을 전달하기 위한 **경로 선택과 중계**를 담당한다.

선지별 해설

① X: 물리 계층에서 받은 비트열을 프레임으로 구분하고 동기를 확인하며, 인접 장치 사이의 흐름 제어와 오류 제어를 수행하는 것은 **데이터 링크 계층**이다. 흐름 제어는 수신 측이 처리할 수 있는 속도에 맞추어 전송량을 조절하고, 오류 제어는 전송 오류를 검출하고 필요한 복구를 수행하는 기능이다. 선지의 '네트워크 계층'을 '데이터 링크 계층'으로 고쳐야 한다. 네트워크 계층은 논리 주소를 바탕으로 목적지까지의 경로를 선택하고 패킷을 중계한다.

② O: 물리 계층은 비트를 신호로 전송하기 위한 **물리적 연결의 설정·유지·해제**를 담당한다. 이를 위해 신호의 전기적 특성, 연결 장치의 기계적 특성 등 실제 전송에 필요한 조건을 규정한다.

③ O: 전송 계층은 송신 측과 수신 측의 **종단 간 데이터 전송**을 담당한다. 필요한 오류 제어와 흐름 제어 등을 통해 상위 계층이 요구하는 전송 품질과 신뢰성을 제공한다. 데이터 링크 계층이 인접 장치 사이의 전송을 다루는 것과 범위가 다르다.

④ O: 응용 계층은 최상위 계층으로, **응용 프로그램이 이용하는 네트워크 서비스**를 제공한다. 사용자가 전자 메일을 보내거나 파일을 전송할 때 필요한 통신 기능과 네트워크 관리 기능 등이 여기에 해당한다.

NoteBase

문 4. <보기> C 프로그램의 실행 결과는?

<보기>

```
#include <stdio.h>

int main( )
{
int a=0, b=1;
switch(a)
    {       case 0 : printf("%d\n", b++); ; break;
            case 1 : printf("%d\n", ++b); ; break;
            default : printf("0\n", b); ; break;
    }
return 0;
}
```

① 0

② 1

③ 2

④ 3

정답 2번

간단한 해설

a가 0이므로 case 0을 실행한다. b++는 **증가 전 값 1을 식의 값으로 사용**하므로 1이 출력되고, break로 switch문을 빠져나간다.

NoteBase

계산 과정

실행 분기: 처음에 a = 0, b = 1이다. switch(a)는 a의 값과 일치하는 case로 이동하므로 **case 0에서 실행을 시작**한다.

후위 증가와 출력: b++는 현재 b의 값을 식의 값으로 사용하면서 b 자체는 1 증가시키는 후위 증가 연산이다. 따라서 printf에 전달되는 값은 **증가 전 값인 1**이고, b에 저장되는 값은 1 + 1 = 2가 된다. %d는 전달받은 정수를 출력하므로 화면에는 1이 출력되고 줄이 바뀐다. 증가한 b의 값 2가 출력되는 것은 아니다.

switch문 종료: 출력문 다음의 **break가 switch문을 종료**하므로 case 1의 ++b와 default의 출력문은 실행되지 않는다. 마지막 return 0은 main 함수를 종료하며, 0을 화면에 출력하는 명령이 아니다.

실행 결과는 1이며, 이는 ②의 값과 일치한다.

NoteBase

문 5. 정책 수립에 있어 중요성이 커지고 있는 빅데이터에 대한 설명으로 가장 옳지 않은 것은?

① 디지털 환경에서 생성되는 데이터로 규모가 방대하고, 생성 주기가 길며, 형태가 다양하다.

② 하둡(Hadoop)과 같은 오픈 소스 소프트웨어 시스템을 빅데이터 처리에 이용하는 것이 가능하다.

③ 보건, 금융과 같은 분야의 빅데이터는 사회적으로 유용한 정보이나 데이터 활용 측면에서 프라이버시 침해에 대한 대비가 필요하다.

④ 구글 및 페이스북, 아마존의 경우 이용자의 성향과 검색 패턴, 구매패턴을 분석해 맞춤형 광고를 제공하는 등 빅데이터의 활용을 증대시키고 있다.

정답 1번

간단한 해설

빅데이터는 데이터가 빠르게 생성·축적되는 **속도(Velocity)**를 주요 특성으로 한다. 따라서 '생성 주기가 길다'는 표현은 **생성 주기가 짧다**로 고쳐야 한다.

선지별 해설

① X: 빅데이터의 대표적인 특성은 방대한 양(Volume), 빠른 생성·처리 속도(Velocity), 다양한 형태(Variety)이다. 이 중 속도는 데이터가 빠르게 생성·축적되고 이를 신속하게 처리할 필요가 있다는 뜻이다. 따라서 '생성 주기가 길다'는 표현은 **생성 주기가 짧다**로 고쳐야 한다. 규모가 방대하고 형태가 다양하다는 부분은 빅데이터의 특성에 해당한다.

② O: 하둡(Hadoop)은 여러 컴퓨터에 데이터를 나누어 저장하고 처리할 수 있도록 하는 오픈 소스 소프트웨어 체계이다. 대량의 데이터를 한 컴퓨터에서 모두 처리하는 부담을 줄이는 **분산 저장·처리**가 가능하므로 빅데이터 처리에 이용할 수 있다.

③ O: 보건 데이터는 질병 연구와 보건 정책에, 금융 데이터는 위험 분석 등에 활용할 수 있다. 그러나 개인의 건강 상태나 거래 내역 같은 민감한 정보가 드러나거나 여러 데이터를 결합해 **개인을 식별할 위험**이 있으므로 접근 통제와 개인정보 보호 조치가 필요하다.

④ O: 구글·페이스북·아마존과 같은 기업이 이용자의 성향, 검색 기록, 구매 패턴 등을 분석하여 맞춤형 광고를 제공하는 것은 빅데이터 활용의 사례이다. 다수 이용자의 행동 데이터에서 관심사를 파악하고 **이용자별로 적합한 광고를 선택**하는 데 분석 결과를 활용한다.

NoteBase

문 6. <보기>는 TCP/IP 프로토콜에 대한 설명이다. ㉠~㉡에 들어갈 내용으로 가장 옳은 것은?

<보기>
· (㉠)는 사용자가 입력한 IP 주소를 이용해 물리적 네트워크 주소(MAC Address)를 제공한다.
· (㉡)는 데이터 전송 과정에서 오류가 발생하면 오류 메시지를 전송한다.

① ㉠ ICMP ㉡ RARP
② ㉠ RARP ㉡ ICMP
③ ㉠ ARP ㉡ ICMP
④ ㉠ ICMP ㉡ ARP

정답 3번

간단한 해설

ARP는 IP 주소에 대응하는 MAC 주소를 알아내는 프로토콜이고, **ICMP**는 IP 패킷 전송 중 발생한 오류 등을 알리는 프로토콜이다. 주소를 알아내는 기능과 전송 오류를 알리는 기능을 구별한다.

빈칸별 해설

㉠ ARP: IP 주소를 알고 있을 때 그 주소에 대응하는 **MAC 주소를 알아낸다**. 이더넷에서 같은 네트워크에 있는 장치로 프레임을 보내려면 목적지의 MAC 주소가 필요하므로, ARP 요청과 응답으로 주소 대응을 확인한다. RARP는 이와 반대로 MAC 주소를 이용해 IP 주소를 알아내는 프로토콜이다.

㉡ ICMP: IP 패킷 전송과 관련된 **오류나 제어 정보를 전달한다**. 예를 들어 목적지에 도달할 수 없거나 패킷의 TTL이 만료되었을 때, 그 사실을 송신 측에 알리는 오류 메시지를 보낼 수 있다. 오류가 난 데이터를 직접 재전송하는 기능과는 구별된다.

㉠에는 ARP, ㉡에는 ICMP가 들어가므로 ③의 조합과 일치한다.

NoteBase

문 7. 주기억 장치의 페이지 교체 기법에 대한 설명으로 가장 옳은 것은?

① FIFO(First In First Out)는 가장 오래된 페이지를 교체한다.

② MRU(Most Recently Used)는 최근에 적게 사용된 페이지를 교체한다.

③ LRU(Least Recently Used)는 가장 최근에 사용한 페이지를 교체한다.

④ LFU(Least Frequently Used)는 최근에 사용빈도가 가장 많은 페이지를 교체한다.

정답 1번

간단한 해설

FIFO는 페이지가 **주기억 장치에 들어온 순서**를 기준으로 교체한다. 현재 적재된 페이지 중 가장 먼저 들어온 페이지를 내보내므로, 이후에 다시 참조되었는지는 교체 순서를 바꾸지 않는다.

선지별 해설

페이지 교체 기법은 새 페이지를 적재할 공간이 부족할 때, 현재 주기억 장치에 있는 페이지 중 어느 것을 내보낼지 정하는 기준이다.

① O: FIFO는 먼저 들어온 페이지를 먼저 내보내는 방식으로, **적재된 지 가장 오래된 페이지**를 교체한다. 예를 들어 A, B 순서로 적재한 뒤 A를 다시 참조해도, 두 페이지 중 교체 대상은 먼저 적재된 A이다.

② X: MRU는 **가장 최근에 사용된 페이지**를 교체한다. Most Recently Used에서 Recently는 사용 시점이 최근이라는 뜻이며, 사용 횟수가 적다는 뜻이 아니다.

③ X: LRU는 마지막으로 사용한 시점이 가장 오래된 페이지, 즉 **가장 오랫동안 사용되지 않은 페이지**를 교체한다. 가장 최근에 사용한 페이지를 교체하는 것은 MRU이다. FIFO가 적재 시점을 기준으로 삼는 데 비해 LRU는 마지막 사용 시점을 기준으로 삼는다.

④ X: LFU는 **사용 빈도가 가장 낮은 페이지**를 교체한다. 페이지별 참조 횟수를 기준으로 사용이 적은 페이지를 내보내는 방식이므로, 사용 빈도가 가장 많은 페이지를 교체한다는 설명은 반대이다.

NoteBase

문 8. RAID (Redundant Array of Inexpensive Disks) 기술에 대한 설명으로 가장 옳지 않은 것은?

① RAID 1 레벨은 미러링(Mirroring)을 지원한다.

② RAID 3 레벨은 데이터를 블록 단위로 분산 저장하여 대용량의 읽기 중심 서버용으로 사용한다.

③ RAID 5 레벨은 고정적인 패리티 디스크 대신 패리티가 모든 디스크에 분산되어 저장되므로 병목 현상을 줄여준다.

④ RAID 6 레벨은 두 개의 패리티 디스크를 사용하므로 두 개의 디스크 장애 시에도 데이터의 복구가 가능하다.

정답 2번

간단한 해설

②의 '블록 단위'가 잘못되었다. RAID 3은 데이터를 **바이트 단위**로 나누어 여러 디스크에 저장하고, 별도의 디스크에 패리티를 저장한다. 블록 단위 분산 저장과 전용 패리티 디스크를 결합한 방식은 RAID 4이다.

선지별 해설

① O: RAID 1은 **동일한 데이터를 다른 디스크에도 복제**하는 미러링 방식이다. 한 디스크에 장애가 생겨도 정상 디스크에 저장된 복사본으로 데이터를 읽을 수 있다.

② X: RAID 3은 데이터를 **바이트 단위로 분산 저장**하고 전용 패리티 디스크를 둔다. 패리티는 디스크 장애 시 데이터를 복구하는 데 쓰는 정보이다. RAID 3은 여러 디스크를 함께 사용하여 대용량 데이터를 순차적으로 읽고 쓰는 데 적합하지만, 분산 단위를 '블록'이라고 한 부분은 틀리다. 블록 단위로 데이터를 분산하고 전용 패리티 디스크를 사용하는 방식은 RAID 4이다.

③ O: RAID 5는 데이터뿐 아니라 **패리티도 여러 디스크에 분산**한다. 전용 패리티 디스크를 두면 쓰기 작업 때 패리티 갱신이 그 디스크에 집중되지만, RAID 5는 패리티 저장 위치를 분산하여 이러한 병목을 줄인다. 패리티를 계산하고 갱신하는 부담 자체가 없어지는 것은 아니다.

④ O: RAID 6은 **서로 독립적인 두 종류의 패리티**를 사용하므로 디스크 두 개가 동시에 고장 나도 남은 데이터와 패리티를 이용해 복구할 수 있다. 선지의 '두 개의 패리티 디스크'는 패리티 저장에 디스크 두 개 분량의 용량을 사용한다는 의미로 이해한다. 실제로는 고정된 두 디스크에만 패리티를 저장하는 것이 아니라 여러 디스크에 분산하여 저장한다.

NoteBase

문 9. 질의 최적화를 위한 질의문의 내부 형태 변화에 대한 규칙으로 가장 옳지 않은 것은?

① 실렉트(select) 연산은 교환적이다: $\sigma_{c_1}(\sigma_{c_2}(R)) \equiv \sigma_{c_2}(\sigma_{c_1}(R))$

② 연속적인 프로젝트(project) 연산은 첫 번째 것을 실행하면 된다:
$\Pi_{List_1}(\Pi_{List_2}(\cdots(\Pi_{List_n}(R))\cdots)) \equiv \Pi_{List_n}(R)$

③ 합집합(∪)과 관련된 프로젝트(project) 연산은 다음과 같이 변환된다: $\Pi(A \cup B) \equiv \Pi(A) \cup \Pi(B)$

④ 실렉트의 조건 c가 프로젝트 속성만 포함하고 있다면 교환적이다: $\sigma_c(\Pi(R)) \equiv \Pi(\sigma_c(R))$

정답 2번

간단한 해설

연속적인 프로젝트 연산은 **마지막에 실행되는 맨 바깥쪽 연산**만 남길 수 있다. ②의 식에서는 최종적으로 남길 속성을 정하는 목록이 $List_1$이므로, 먼저 실행되는 $List_n$의 프로젝트만 남겨서는 안 된다.

선지별 해설

실렉트는 조건을 만족하는 행을 고르고, 프로젝트는 지정된 속성에 해당하는 열을 남기는 연산이다. 질의 변환 규칙은 연산식을 바꾸어도 결과가 같다는 성질을 이용한다.

① O: 두 실렉트를 연속 적용하면 두 조건을 모두 만족하는 행만 남는다. 조건을 적용하는 순서를 바꾸어도 **두 조건을 모두 만족해야 한다**는 기준은 같으므로 결과가 변하지 않는다.

② X: 중첩된 연산은 안쪽부터 실행되므로, 먼저 실행되는 것은 가장 안쪽 프로젝트이고 마지막에 실행되는 것은 가장 바깥쪽 프로젝트이다. 각 단계에서 바깥쪽 속성 목록이 바로 안쪽 속성 목록에 포함되어 연산이 성립한다면, **맨 바깥쪽 프로젝트만 남긴다**. 따라서 제시된 식의 우변은 $\Pi_{List_1}(R)$로 고쳐야 한다. 예를 들어 이름·학과를 남긴 뒤 이름만 남기는 결과는 처음부터 이름만 남기는 결과와 같다. 이름·학과를 남기는 첫 연산만 실행하면 최종 결과에 불필요한 학과가 남는다.

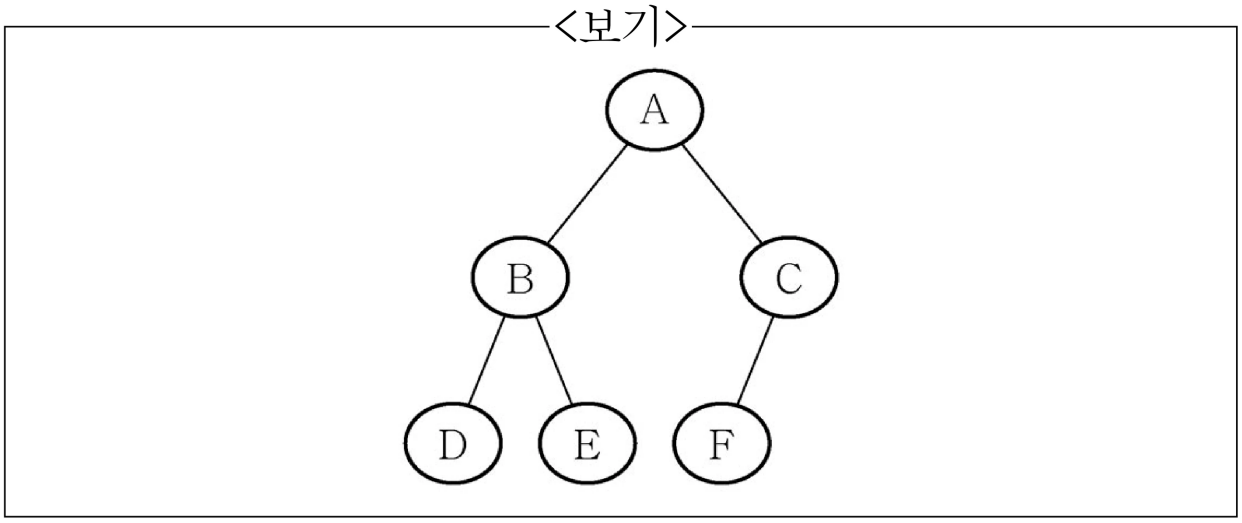
③ O: 합집합이 가능한 두 릴레이션에서 **같은 속성 목록**으로 프로젝트한다면, 합친 뒤 해당 열을 남기든 각 릴레이션에서 해당 열을 남긴 뒤 합치든 결과가 같다. 어느 쪽이든 두 릴레이션에 나타난 해당 속성값들을 모은 집합이 되므로 프로젝트는 합집합에 대해 분배된다.

④ O: 프로젝트를 먼저 해도 실렉트를 적용하려면 **조건 c에서 쓰는 모든 속성이 남아 있어야 한다**. 이 조건에서는 열을 줄인 뒤 행을 고르거나 행을 고른 뒤 열을 줄여도 같은 결과가 나온다. 반대로 조건에 필요한 속성을 프로젝트가 제거하면, 이후 그 조건을 평가할 수 없으므로 두 연산을 그대로 교환할 수 없다.

NoteBase

문 10. <보기> 이진 트리의 내부 경로 길이(length)와 외부 경로 길이로 옳은 것은?

<보기>



① 7, 20

② 7, 23

③ 8, 20

④ 8, 23

정답 3번

간단한 해설

루트의 깊이를 **0**으로 두고 A~F의 깊이를 합하면 내부 경로 길이는 8이다. 외부 경로 길이는 **빈 자식 자리에 추가한 외부 노드**의 깊이를 합하므로, 깊이 2인 노드 1개와 깊이 3인 노드 6개를 합산한 20이다.

NoteBase

계산 과정

노드의 깊이는 루트에서 해당 노드까지 거치는 간선의 수이다. 내부·외부 경로 길이는 각 대상 노드의 깊이를 모두 더한 값이다. 빈 자식 자리마다 외부 노드를 추가한 확장 이진 트리를 기준으로 하므로, 그림의 A~F는 모두 내부 노드에 해당한다.

내부 경로 길이: A의 깊이는 0, B·C의 깊이는 각각 1, D·E·F의 깊이는 각각 2이다. 따라서 0 + 1 + 1 + 2 + 2 + 2 = 8이다. 자식이 없는 **D·E·F도 기존 노드이므로 합산**한다.

외부 경로 길이: C의 오른쪽 자식이 비어 있으므로 그 자리에 깊이 2인 외부 노드 1개를 추가한다. D·E·F는 각각 두 자식이 모두 비어 있으므로 깊이 3인 외부 노드를 총 6개 추가한다. 이처럼 **빈 자식 자리 각각을 따로 세어** 깊이를 더하면 2 + 6 × 3 = 20이다.

내부 경로 길이와 외부 경로 길이는 각각 8과 20으로, ③의 값과 일치한다.

NoteBase

문 11. 8진수로 표현된 $13754_{(8)}$를 10진수로 표현하면?

① 6224

② 6414

③ 6244

④ 6124

정답 4번

간단한 해설

8진수는 오른쪽 끝자리부터 **1, 8, 64, 512, 4096의 자리값**을 적용한다. 각 숫자에 해당 자리값을 곱해 더하면 10진수 6124가 된다.

계산 과정

8진수는 한 자리 왼쪽으로 이동할 때마다 자리값이 **8배**가 된다. 따라서 $13754_{(8)}$의 각 자리에는 왼쪽부터 $8^4, 8^3, 8^2, 8^1, 8^0$을 곱한다.

각 자리가 나타내는 값을 계산하면 다음과 같다.

$1 \cdot 8^4 = 1 \cdot 4096 = 4096$

$3 \cdot 8^3 = 3 \cdot 512 = 1536$

$7 \cdot 8^2 = 7 \cdot 64 = 448$

$5 \cdot 8^1 = 5 \cdot 8 = 40$

$4 \cdot 8^0 = 4 \cdot 1 = 4$

이 값들을 더하면 4096 + 1536 = 5632이고, 5632 + 448 + 40 + 4 = 6124이다. 따라서 10진수 표현은 6124이며, ④에 해당한다.

NoteBase

문 12. <보기> 잘 알려진 포트번호(well-known port)와 TCP 프로토콜이 바르게 연결된 것을 모두 고른 것은?

<보기>
ㄱ: 21번 포트 : FTP
ㄴ: 53번 포트 : TELNET
ㄷ: 23번 포트 : SMTP
ㄹ: 80번 포트 : HTTP

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄴ, ㄹ

정답 2번

간단한 해설

FTP의 제어 연결은 **TCP 21번**을, HTTP의 기본 연결은 **TCP 80번**을 사용한다.

보기별 해설

ㄱ O: FTP는 파일 전송에 사용하는 프로토콜이다. 명령과 응답을 주고받는 **제어 연결에 TCP 21번**을 사용하므로 바르게 연결되었다.

ㄴ X: 53번은 도메인 이름을 IP 주소 등으로 조회하는 **DNS의 포트**이다. 원격 접속 프로토콜인 TELNET의 기본 포트는 TCP 23번이므로, TELNET과 연결하려면 53번을 23번으로 고쳐야 한다.

ㄷ X: TCP 23번은 TELNET의 기본 포트이다. 전자우편 전송 프로토콜인 **SMTP의 기본 포트는 TCP 25번**이므로, SMTP와 연결하려면 23번을 25번으로 고쳐야 한다.

ㄹ O: HTTP는 웹 클라이언트와 서버 사이에서 요청과 응답을 주고받는 프로토콜이다. **기본 포트가 TCP 80번**이므로 바르게 연결되었다.

바르게 연결된 보기는 ㄱ과 ㄹ이며, 이는 ②의 조합과 일치한다.

NoteBase

문 13. 파일 처리 시스템(File Process System)과 비교한 데이터 베이스관리 시스템(DBMS)에 대한 설명으로 가장 옳지 않은 것은?

① 응용 프로그램과 데이터 간의 상호 의존성이 크다.

② 데이터 중복을 최소화한다.

③ 응용 프로그램의 요청을 수행한다.

④ 데이터 공유를 수월하게 한다.

정답 1번

간단한 해설

①의 '상호 의존성이 크다'는 파일 처리 시스템의 특성이다. DBMS는 데이터의 저장과 접근을 담당하여 응용 프로그램과 데이터 간의 의존성을 줄이는 **데이터 독립성**을 제공한다.

선지별 해설

① X: 파일 처리 시스템에서는 응용 프로그램이 파일 구조에 맞추어 데이터를 직접 처리하므로, 파일 구조가 바뀌면 프로그램도 수정해야 하는 경우가 많다. DBMS는 응용 프로그램과 데이터 사이에서 저장과 접근을 관리하여 이러한 의존성을 줄인다. 따라서 '상호 의존성이 크다'는 **'데이터 독립성이 높다'**로 고쳐야 한다.

② O: DBMS는 여러 프로그램이 따로 보관하던 데이터를 통합하여 관리하므로 **같은 데이터의 불필요한 중복**을 줄일 수 있다. 이는 저장 공간의 낭비와 중복된 값이 서로 달라지는 문제를 줄이는 데 도움이 된다. 중복을 최소화한다는 뜻이지 모든 중복을 반드시 없앤다는 뜻은 아니다.

③ O: DBMS는 응용 프로그램이 보낸 **데이터 조회·삽입·수정·삭제 요청**을 처리한다. 응용 프로그램이 필요한 데이터를 요청하면 DBMS가 데이터베이스에 접근하여 해당 작업을 수행하고 결과를 돌려준다.

④ O: DBMS는 여러 사용자와 응용 프로그램이 **공통의 데이터베이스를 이용**할 수 있게 한다. 접근 권한을 관리하고 동시에 이루어지는 작업을 제어하므로, 프로그램마다 별도 파일을 유지하는 방식보다 데이터를 공유하기 쉽다.

NoteBase

문 14. 임계지역(critical section) 문제에 대한 해결책이 가져야 하는 성질로 가장 옳지 않은 것은?

① 한 번에 한 프로세스만이 임계지역을 수행하도록 해야 한다.

② 프로세스는 자신이 임계지역을 수행하지 않으면서 다른 프로세스가 임계지역을 수행하는 것을 막으면 안된다.

③ 프로세스의 임계지역 진입은 유한 시간 내에 이루어져야 한다.

④ 임계지역 문제의 해결책에서는 프로세스의 수행 속도에 대해 적절한 가정을 할 수 있다.

정답 4번

간단한 해설

④의 '수행 속도에 대해 적절한 가정을 할 수 있다'는 설명이 잘못되었다. 임계지역 문제의 해결책은 **프로세스의 상대적인 수행 속도에 의존하지 않고** 올바르게 동작해야 한다.

선지별 해설

① O: 상호 배제에 관한 설명이다. 임계지역은 공유 자료에 접근하는 코드 부분으로, 한 프로세스가 실행 중이면 다른 프로세스는 **같은 공유 자료의 임계지역에 동시에 진입할 수 없어야** 한다. 그래야 서로의 갱신을 덮어쓰는 등의 오류를 막을 수 있다.

② O: 진행에 관한 설명이다. 임계지역이 비어 있고 진입하려는 프로세스가 있으면, 다음에 들어갈 프로세스를 결정하는 일이 무한히 미뤄져서는 안 된다. 특히 **진입 의사가 없는 프로세스가 다른 프로세스의 진입을 막아서는 안 된다**. 예를 들어 A가 임계지역에 들어갈 생각이 없는데도 A의 차례라는 이유로 B를 계속 기다리게 하면 진행 조건을 위반한다.

③ O: 한정 대기에 관한 설명이다. 진입을 요청한 프로세스가 다른 프로세스에 계속 밀려 무한히 기다리는 기아 상태를 막아야 한다. 구체적으로는 요청 후 자신이 진입하기까지 **다른 프로세스가 먼저 진입할 수 있는 횟수에 한계**가 있어야 한다. 이는 일정한 초 단위의 대기 시간을 미리 정한다는 뜻은 아니다.

④ X: 임계지역 문제의 해결책은 **프로세스 간 상대적인 수행 속도를 가정해서는 안 된다**. 어느 프로세스가 더 빠른지에 기대어 진입을 제어하면 실행 속도나 실행 순서가 달라졌을 때 상호 배제 등이 깨질 수 있다. 따라서 '적절한 가정을 할 수 있다'를 '상대적인 수행 속도에 대한 가정 없이 동작해야 한다'로 고쳐야 한다.

NoteBase

문 15. <보기> C 프로그램의 출력은?

<보기>

```
#include <stdio.h>

int main()
{
    int a = 5, b = 5;

    a *= 3 + b++;
    printf("%d %d", a, b) ;
    return 0;
}
```

① 40 5

② 40 6

③ 45 5

④ 45 6

정답 2번

간단한 해설

b++는 식에서 **증가 전 값 5**를 사용하고 b를 6으로 만든다. a에는 5 × (3 + 5) = 40이 저장되므로, 출력할 때 a는 40, b는 6이다.

NoteBase

계산 과정

초깃값: int a = 5, b = 5에 따라 두 정수 변수 a와 b는 모두 5에서 시작한다.

후위 증가: b++는 **현재 값을 식에 제공한 뒤 b를 1 증가**시키는 연산이다. 따라서 3 + b++에서 덧셈에 사용하는 값은 5이고, 오른쪽 식의 값은 3 + 5 = 8이다. 이 문장의 실행이 끝나면 b는 6이다.

복합 대입: a *= 3 + b++는 **오른쪽 식 전체의 값을 기존 a에 곱해 저장**한다. 오른쪽 식의 값이 8이므로 a에는 5 × 8 = 40이 저장된다. b가 6이 되었다고 해서 이미 덧셈에 사용한 값 5가 바뀌지는 않는다.

출력: printf의 두 %d는 a와 b를 차례로 정수 형식으로 출력하며, 그 사이에는 공백이 들어간다. 출력은 '40 6'이며 ②와 일치한다.

NoteBase

문 16. <보기> 회로의 종류를 바르게 연결한 것은?

<보기>
ㄱ. 3개의 입력 중에서 적어도 2개의 입력이 1이면 출력이 1이 되는 회로
ㄴ. 설정된 값이 표시되었을 때, 경고음을 울리는 카운터

① ㄱ: 조합논리회로, ㄴ: 조합논리회로
② ㄱ: 조합논리회로, ㄴ: 순차논리회로
③ ㄱ: 순차논리회로, ㄴ: 조합논리회로
④ ㄱ: 순차논리회로, ㄴ: 순차논리회로

정답 2번

간단한 해설

ㄱ은 **현재 입력의 조합만으로 출력이 결정**된다. 반면 ㄴ의 카운터는 입력을 셀 때 **이전까지의 계수값을 기억**해야 하므로 순차논리회로에 해당한다.

보기별 해설

ㄱ. 조합논리회로: 출력이 현재 입력값만으로 결정되는 회로이다. 주어진 회로는 세 입력 중 1의 개수만 확인하면 된다. 예를 들어 입력이 1·1·0이면 출력은 1이고, 1·0·0이면 출력은 0이다. **이전에 어떤 입력이 들어왔는지 기억할 필요가 없으므로** 조합논리회로이다.

ㄴ. 순차논리회로: 현재 입력뿐 아니라 저장된 이전 상태에도 동작이 의존하는 회로이다. 카운터는 입력 펄스가 들어오면 **기존 계수값을 바탕으로 값을 갱신**하고, 표시된 값이 설정값에 도달하면 경고음을 울린다. 설정값과 현재 값의 비교 자체는 조합논리로 구현할 수 있지만, 계수값을 기억하고 갱신하는 카운터 전체는 순차논리회로이다.

ㄱ은 조합논리회로, ㄴ은 순차논리회로이므로 ②의 연결과 일치한다.

NoteBase

문 17. CISC(Complex Instruction Set Computer)에 대한 설명으로 가장 옳은 것은?

① 고정 길이의 명령어 형식을 가진다.

② 명령어의 길이가 짧다.

③ 다양한 어드레싱 모드를 사용한다.

④ 하나의 명령으로 복잡한 명령을 수행할 수 없어 복잡한 하드웨어가 필요하다.

정답 3번

간단한 해설

CISC는 피연산자의 위치를 지정하는 **다양한 어드레싱 모드**를 지원한다. 명령어가 레지스터나 메모리 등 여러 위치의 데이터를 다룰 수 있도록 설계된 방식이다.

선지별 해설

① X: CISC는 일반적으로 **가변 길이 명령어**를 사용한다. 연산의 종류와 피연산자 지정 방식에 따라 필요한 비트 수가 달라지기 때문이다. 고정 길이의 단순한 명령어 형식은 전형적인 RISC의 특징이다.

② X: CISC의 명령어는 다양한 연산과 주소 지정 정보를 표현하므로 길이가 일정하게 짧다고 볼 수 없다. 전형적인 비교에서는 CISC를 **길이가 다양하고 상대적으로 복잡한 명령어**를 사용하는 방식으로, RISC를 짧고 단순한 명령어를 사용하는 방식으로 구분한다.

③ O: 어드레싱 모드는 명령어가 사용할 피연산자나 그 위치를 지정하는 방식이다. CISC는 **다양한 어드레싱 모드**를 지원한다. 예를 들어 명령어 안의 상수를 직접 사용하거나, 레지스터 또는 메모리에 있는 데이터를 지정할 수 있다.

④ X: CISC는 **하나의 명령어로 복잡한 작업을 수행할 수 있다**. 다양한 명령어와 주소 지정 방식을 해석하고 실행하도록 제어장치 등 하드웨어가 복잡해지는 경향이 있다. 따라서 '복잡한 명령을 수행할 수 없어'를 '복잡한 작업을 수행할 수 있도록'으로 고쳐야 한다.

NoteBase

문 18. 퀵 정렬에 대한 설명으로 가장 옳지 않은 것은?

① 퀵 정렬은 분할 정복(divide and conquer) 방식으로 동작한다.

② 퀵 정렬의 구현은 흔히 재귀 함수 호출을 포함한다.

③ n개의 데이터에 대한 퀵 정렬의 평균 수행 시간은 $O(\log n)$이다.

④ C.A.R. Hoare가 고안한 정렬 방식이다.

정답 3번

간단한 해설

퀵 정렬의 평균 수행 시간은 $O(n\log n)$이다. 분할이 대체로 균형을 이루면 분할 단계가 약 $\log n$개이지만, **각 단계에서 전체적으로 n개 정도의 데이터를 비교·분할하는 작업**도 필요하다.

선지별 해설

① O: 퀵 정렬은 피벗을 기준으로 데이터를 나누고, 나뉜 부분에 같은 정렬 방법을 적용하는 **분할 정복** 알고리즘이다. 각 부분의 정렬이 끝나면 전체 데이터도 정렬된다.

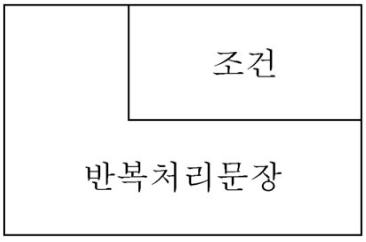
② O: 분할된 작은 구간에도 같은 정렬 절차를 적용하므로 흔히 **재귀 함수 호출**로 구현한다. 각 구간을 다시 분할하고 정렬하는 함수를 호출하며, 구간의 데이터가 0개 또는 1개이면 더 정렬할 필요가 없어 재귀 호출을 끝낸다.

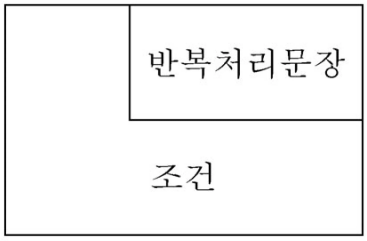
③ X: 데이터가 n개일 때 퀵 정렬의 평균 수행 시간은 $O(\log n)$이 아니라 $O(n\log n)$이다. 한 구간을 피벗 기준으로 분할하려면 그 구간의 데이터를 살펴야 한다. 분할이 대체로 균형을 이루는 경우 단계 수는 약 $\log n$이고, 같은 단계의 여러 구간을 합치면 전체적으로 n개 정도의 데이터를 처리하므로 평균 수행 시간에 n이라는 요인이 포함된다.

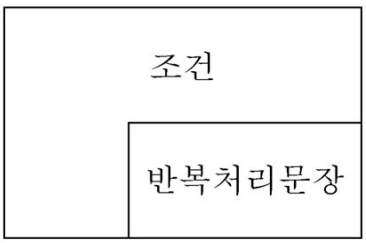
④ O: 퀵 정렬을 고안한 사람은 **C.A.R. Hoare**이다.

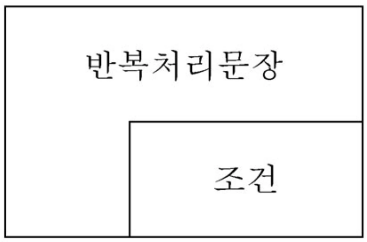
NoteBase

문 19. 나시-슈나이더만(N-S) 차트의 반복(While) 구조에 대한 표현으로 가장 옳은 것은?

① 

② 

③ 

④ 

정답 3번

간단한 해설

While은 **조건을 먼저 검사**하여 참일 때만 반복 처리문을 실행한다. N-S 차트에서는 조건 영역이 반복 처리문의 위쪽과 왼쪽을 감싸고, 반복 처리문은 오른쪽 아래의 내부 칸에 놓인다.

NoteBase

도식 해설

While은 매번 반복 처리문을 실행하기 전에 조건을 검사하는 선검사 반복 구조이다. 조건이 참이면 처리문을 실행한 뒤 다시 조건을 검사하고, 거짓이면 반복을 끝낸다. 처음부터 조건이 거짓이면 처리문을 한 번도 실행하지 않는다. 도식에서는 조건 영역이 위쪽에서 왼쪽으로 이어져 반복 처리문을 감싸는지를 확인한다.

①: '조건'이 오른쪽 위의 내부 칸에 있고, '반복처리문장'이 바깥쪽 영역에 있다. While에서는 이와 달리 **반복 처리문이 오른쪽 아래의 내부 칸**에 들어가야 한다.

②: 반복 처리문이 오른쪽 위의 내부 칸에 있고, 조건 영역이 아래쪽에서 왼쪽으로 이어져 처리문을 감싼다. 이는 **처리문 실행 후 조건 검사**를 하는 후검사 반복의 배치이다. 후검사 반복은 처리문을 최소 한 번 실행하므로, 처음부터 조건이 거짓이면 실행하지 않는 While과 구별된다.

③: 조건 영역이 위쪽과 왼쪽으로 이어지고, 반복 처리문이 오른쪽 아래의 내부 칸에 들어 있다. 이는 **조건 검사 후 처리문 실행**이라는 While의 선검사 반복 구조를 나타낸다.

④: 바깥쪽 위·왼쪽 영역에 '반복처리문장'이, 오른쪽 아래의 내부 칸에 '조건'이 있다. While 도식과 비교하면 **조건과 반복 처리문의 위치가 서로 바뀌어** 있다. While은 바깥쪽 위·왼쪽 영역에 조건을 두고, 오른쪽 아래의 내부 칸에 반복 처리문을 두어야 한다.

NoteBase

문 20. <보기> C 프로그램의 실행 결과로 화면에 출력되는 숫자가 아닌 것은?

<보기>

```
#include <stdio.h>

int my(int i, int j) {
    if (i<3) i=j=1;
    else {
       i=i-1;
       j=j-i;
       printf("%d, %d, ", i, j);
       return my(i,j);
    }
}

int main(void)
{
     my(5,14);
     return 0;
}
```

① 1

② 3

③ 5

④ 7

정답 1번

간단한 해설

i가 3보다 작으면 i와 j를 1로 바꾸지만, 출력문은 **else 블록 안에만** 있다. 따라서 종료 분기에서 대입한 1은 화면에 출력되지 않는다.

NoteBase

계산 과정

main에서 my(5,14)를 호출하므로 처음 i는 5, j는 14이다. my는 i < 3이면 i와 j에 1을 대입하고, 그렇지 않으면 i와 j를 갱신하여 출력한 뒤 자신을 다시 호출한다. 이때 **j를 계산할 때는 이미 감소한 i**를 사용한다.

첫 호출 my(5,14): i < 3이 거짓이므로 else 블록을 실행한다. i = 5 − 1 = 4, j = 14 − 4 = 10이 되어 4, 10을 출력하고 my(4,10)을 호출한다.

다음 호출 my(4,10): 다시 else 블록을 실행한다. i = 4 − 1 = 3, j = 10 − 3 = 7이 되어 3, 7을 출력하고 my(3,7)을 호출한다.

다음 호출 my(3,7): 조건을 검사할 때 i는 3이므로 i < 3은 거짓이다. 따라서 else 블록에서 i = 3 − 1 = 2, j = 7 − 2 = 5로 바꾸고 2, 5를 출력한 뒤 my(2,5)를 호출한다. **i가 2가 된 직후에도 출력문은 실행**되며, 조건은 다음 호출에서 다시 검사한다.

마지막 호출 my(2,5): i < 3이 참이므로 i = j = 1을 실행한다. 이는 j와 i에 모두 1을 대입하는 문장이다. 이 분기에는 **출력문과 재귀 호출이 없으므로** 1을 출력하지 않고 재귀 호출을 끝낸다.

화면에 출력되는 숫자는 순서대로 4, 10, 3, 7, 2, 5이다. 따라서 선택지의 3, 5, 7은 출력되지만, ①의 1은 출력되지 않는다.

NoteBase