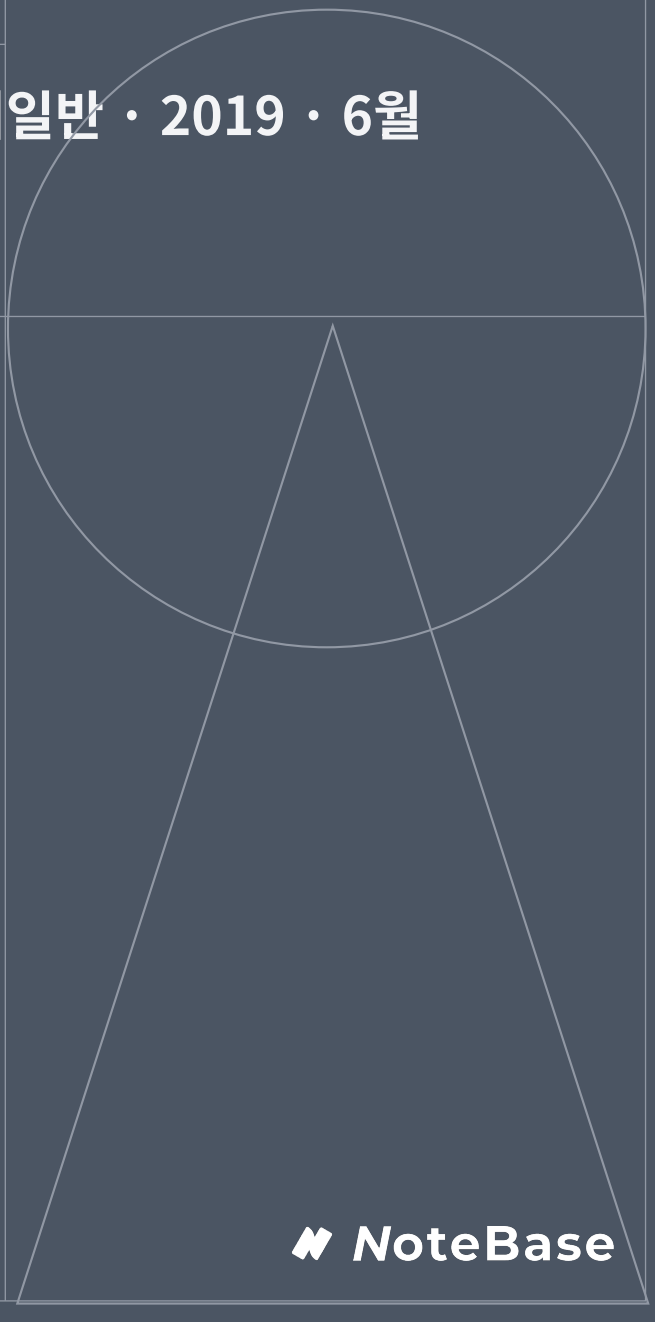
# 기출OX 해설집

공무원/서울시/9급 · 컴퓨터일반 · 2019 · 6월



NoteBase

# 기출OX 해설집

## 공무원/서울시/9급 · 컴퓨터일반 · 2019 · 6월

**공무원/서울시/9급 · 컴퓨터일반 · 2019 · 6월**

**문 1. C 프로그램을 컴파일하면 <보기>와 같은 것들이 실행된다. 이 중 3번째로 실행되는 것은?**

<보기>
링커(linker), 어셈블러(assembler),
전처리기(preprocessor), 컴파일러(compiler)

① 링커(linker)
② 어셈블러(assembler)
③ 전처리기(preprocessor)
④ 컴파일러(compiler)

**정답 2번**

**간단한 해설**

전처리로 정리한 C 코드는 컴파일러를 거쳐 어셈블리 코드가 되고, 이어 **어셈블러가 목적 코드로 변환**한다.

**올바른 순서**

전처리기 → 컴파일러 → 어셈블러 → 링커

전처리기: #include에 지정된 파일의 내용을 포함하고 #define으로 정의한 매크로를 치환하는 등 **컴파일에 앞서 소스 코드를 처리**한다. 이렇게 정리된 C 코드가 컴파일러의 입력이 된다.

컴파일러: 전처리된 C 코드를 분석하여 **어셈블리 코드로 번역**한다. 이 결과가 다음 단계인 어셈블러의 입력이 된다.

어셈블러: 컴파일러가 생성한 어셈블리 코드를 **기계어로 된 목적 코드로 변환**한다. 전처리기와 컴파일러 다음에 실행되므로 세 번째 단계이다.

링커: 어셈블러가 만든 목적 파일과 필요한 라이브러리 등을 연결하고, 파일 사이의 외부 기호 참조를 해결하여 **실행 파일을 생성**한다. 연결할 목적 파일이 먼저 만들어져야 하므로 어셈블러 다음에 실행된다.

NoteBase

**문 2. 유닉스 파일 시스템에 대한 설명으로 가장 옳지 않은 것은?**

① 슈퍼블록은 전체 블록의 수, 블록의 크기, 사용 중인 블록의 수 등 파일 시스템의 정보를 가지고 있다.

② 아이노드는 파일의 종류, 크기, 소유자, 접근 권한 등 각종 속성 정보를 가지고 있다.

③ 파일마다 데이터 블록, 아이노드 외에 직접 블록 포인터와 단일·이중·삼중 간접 블록 포인터로 구성된 인덱스 정보를 가진 인덱스 블록을 별도로 가지고 있다.

④ 디렉터리는 하위 파일들의 이름과 아이노드 포인터(또는 아이노드 번호)를 포함하는 디렉터리 엔트리들로 구성된다.

**정답 3번**

**간단한 해설**

직접 블록 포인터와 단일·이중·삼중 간접 블록 포인터는 별도의 인덱스 블록이 아니라 **아이노드 안에** 들어 있다. 간접 블록은 파일 크기에 따라 필요할 때 사용하며, 추가적인 블록 주소를 저장한다.

**선지별 해설**

① O: 슈퍼블록은 전체 블록 수, 블록 크기, 블록의 사용·여유 현황 등 **파일 시스템 전체의 관리 정보**를 담는다. 개별 파일의 속성을 담는 아이노드와 관리 범위가 다르다.

② O: 아이노드는 **개별 파일의 속성 정보**를 저장하는 구조이다. 파일의 종류와 크기, 소유자, 접근 권한 등을 통해 파일을 관리하며, 파일 데이터가 저장된 블록을 찾는 주소 정보도 담는다.

③ X: 직접 블록 포인터와 단일·이중·삼중 간접 블록 포인터는 **아이노드 내부에 저장**된다. 직접 포인터는 데이터 블록을 바로 가리키고, 간접 포인터는 블록 주소들을 담은 간접 블록을 거쳐 데이터 블록에 도달한다.
단일·이중·삼중은 각각 간접 블록을 한 단계·두 단계·세 단계 거친다는 뜻이다. 작은 파일은 직접 포인터만으로 처리할 수 있으므로, 모든 파일이 이 포인터들을 한데 모은 별도의 인덱스 블록을 갖는다는 설명은 잘못이다.

④ O: 디렉터리는 **파일 이름과 아이노드 번호를 연결**하는 디렉터리 엔트리들로 구성된다. 파일 이름으로 해당 엔트리를 찾고, 그 아이노드 번호를 이용하여 파일의 속성과 데이터 블록 주소에 접근한다.

NoteBase

**문 3. <보기>는 8비트에 부호 있는 2의 보수 표현법으로 작성한 이진수이다. 이에 해당하는 십진 정수는?**

<보기>
10111100

① −60
② −68
③ 94
④ 188

**정답 2번**

**간단한 해설**

8비트 2의 보수 표현에서는 최상위 비트 1의 가중치가 **−128**이다. 나머지 7비트의 값 60을 더하면 −68이 된다.

**계산 과정**

최상위 비트의 가중치: 8비트 2의 보수 표현에서 최상위 비트는 음의 가중치 $-2^7 = -128$을 갖는다.
10111100은 **최상위 비트가 1이므로 음수**이며, 이 비트가 나타내는 값은 −128이다.

나머지 비트의 값: 최상위 비트를 제외한 0111100의 가중치는 왼쪽부터 64, 32, 16, 8, 4, 2, 1이다. 이 중 비트가 1인 자릿값을 더하면 32 + 16 + 8 + 4 = 60이다. 이 60은 전체 수의 절댓값이 아니라, **하위 7비트가 기여하는 양의 값**이다.

십진 정수로 변환: 최상위 비트의 값과 하위 7비트의 값을 합하면 −128 + 60 = −68이다. 따라서 주어진 이진수의 십진 정수는 −68이며, ②의 값과 일치한다.

NoteBase

**문 4. <보기>가 설명하는 것은?**

<보기>
다음에 실행할 명령어의 주소를 보관하는 레지스터이다. 계수기로 되어 있어 실행할 명령어를 메모리에서 읽으면 명령어의 길이만큼 증가하여 다음 명령어를 가리키며, 분기 명령어는 목적 주소로 갱신할 수 있다.

① 명령어 레지스터
② 프로그램 카운터
③ 데이터 레지스터
④ 주소 레지스터

**정답 2번**

**간단한 해설**

다음에 실행할 명령어의 주소를 보관하고, 순차 실행 때는 명령어 길이만큼 증가하며 분기 때는 목적 주소로 갱신되는 레지스터는 **프로그램 카운터(PC)**이다.

**공통 해설**

프로그램 카운터(PC)는 CPU가 다음에 가져와 실행할 명령어의 메모리 주소를 보관한다. 핵심은 단순히 주소를 저장한다는 점이 아니라 **다음 명령어를 가져올 위치를 지정**한다는 점이다. 순차 실행에서는 명령어를 읽은 뒤 그 길이만큼 PC가 증가하고, 분기가 일어나면 분기 목적 주소가 PC에 저장된다.

예를 들어 바이트 단위 주소를 사용하는 컴퓨터에서 주소 100에 있는 4바이트 명령어를 읽었다면, 순차 실행을 위한 PC 값은 104가 된다. 이 명령어의 실행으로 주소 200으로 분기하면 PC는 200으로 바뀐다. 이처럼 PC의 갱신은 다음에 실행할 명령어의 위치를 결정한다.

NoteBase

**문 5. 운영체제에서 가상 메모리의 페이지 교체 기법에 대한 설명으로 가장 옳지 않은 것은?**

① FIFO 기법에서는 아무리 참조가 많이 된 페이지라도 교체될 수 있다.

② LRU 기법을 위해서는 적재된 페이지들의 참조된 시간 또는 순서에 대한 정보가 필요하다.

③ Second-chance 기법에서는 참조 비트가 0인 페이지는 교체되지 않는다.

④ LFU 기법은 많이 참조된 페이지는 앞으로도 참조될 확률이 높을 것이란 판단에 근거한 기법이다.

**정답 3번**

**간단한 해설**

Second-chance 기법은 교체 후보의 **참조 비트가 0이면 그 페이지를 교체**한다. 참조 비트가 1이면 0으로 바꾸고 한 번의 기회를 더 준 뒤 다음 후보를 확인한다.

**선지별 해설**

① O: FIFO는 **메모리에 먼저 적재된 페이지**를 먼저 교체한다. 적재 후의 참조 횟수는 교체 기준이 아니므로, 여러 번 참조된 페이지도 가장 먼저 적재되었다면 교체될 수 있다.

② O: LRU는 **마지막 참조 이후 가장 오래 지난 페이지**를 교체한다. 이를 판단하려면 적재된 페이지가 마지막으로 참조된 시간이나 페이지 사이의 최근 참조 순서를 알아야 한다.

③ X: Second-chance는 FIFO 방식의 교체 순서에 참조 비트를 추가하여 최근에 참조된 페이지에 한 번의 기회를 더 주는 기법이다. 교체 후보를 확인했을 때 **참조 비트가 0이면 교체**하고, 1이면 비트를 0으로 바꾼 뒤 다음 후보로 넘어간다. 따라서 '참조 비트가 0인 페이지는 교체되지 않는다'는 설명은 '참조 비트가 0인 페이지는 교체 대상으로 선택된다'로 고쳐야 한다.

④ O: LFU는 **참조 횟수가 가장 적은 페이지**를 교체한다. 과거에 많이 참조된 페이지는 앞으로도 참조될 가능성이 높다고 보고 메모리에 남기려는 방식이다. LRU가 마지막 참조 시점을 기준으로 삼는 데 비해, LFU는 누적 참조 횟수를 기준으로 삼는다.

NoteBase

**문 6. 네트워킹 장비에 대한 설명으로 가장 옳지 않은 것은?**

① 라우터(router)는 데이터 전송을 위한 최선의 경로를 결정한다.

② 허브(hub)는 전달받은 신호를 그와 케이블로 연결된 모든 노드들에 전달한다.

③ 스위치(switch)는 보안(security) 및 트래픽(traffic) 관리 기능도 제공할 수 있다.

④ 브리지(bridge)는 한 네트워크 세그먼트에서 들어온 데이터를 그의 물리적 주소에 관계없이 무조건 다른 세그먼트로 전달한다.

**정답 4번**

**간단한 해설**

브리지는 **목적지 MAC 주소에 따라** 프레임을 전달하거나 걸러 낸다. 목적지가 수신한 쪽과 같은 세그먼트에 있으면 다른 세그먼트로 전달하지 않으므로, '물리적 주소에 관계없이 무조건 전달한다'는 설명은 틀리다.

**선지별 해설**

① O: 라우터는 서로 다른 네트워크를 연결하며, 패킷의 목적지 IP 주소와 라우팅 테이블을 바탕으로 다음 전달 경로를 선택한다. 라우팅에서 정한 비용 등의 기준에 따라 **최선의 경로**를 결정한다.

② O: 허브는 물리 계층에서 동작하며, 한 포트로 받은 신호를 **수신 포트를 제외한 다른 모든 포트**로 내보낸다. 목적지 주소를 분석해 특정 노드만 골라 전달하지 않으므로, 연결된 다른 노드들도 신호를 받는다.

③ O: 스위치는 MAC 주소를 바탕으로 프레임을 전달한다. 관리 기능을 갖춘 스위치는 포트별 접속을 제한하는 보안 기능이나 트래픽 우선순위를 지정하는 관리 기능도 제공할 수 있다. 따라서 이런 기능을 **제공할 수 있다**는 설명이 성립한다.

④ X: 브리지는 데이터 링크 계층에서 동작하며, 물리적 주소인 **MAC 주소를 기준으로 전달 여부**를 결정한다. 학습한 주소 정보를 조회하여 목적지가 다른 세그먼트에 있으면 그쪽으로 전달하고, 수신한 쪽과 같은 세그먼트에 있으면 다른 세그먼트로 전달하지 않는다. 예를 들어 같은 세그먼트의 A가 B에게 보낸 프레임은 B의 위치를 알고 있는 브리지를 통해 다른 세그먼트로 넘어가지 않는다. 목적지 위치를 아직 모르면 수신한 쪽을 제외한 다른 연결 세그먼트로 전달할 수 있지만, 모든 프레임을 주소와 관계없이 무조건 전달하는 것은 아니다.

NoteBase

**문 7. 다음의 정렬된 데이터에서 2진탐색을 수행하여 C를 찾으려고 한다. 몇 번의 비교를 거쳐야 C를 찾을 수 있는가? (단, 비교는 ‘크다’, ‘작다’, ‘같다’ 중의 하나로 수행되고, ‘같다’가 도출될 때까지 반복된다.)**

A B C D E F G H I J K L M N O

① 1번

② 2번

③ 3번

④ 4번

**정답 4번**

**간단한 해설**

C가 탐색 범위의 유일한 원소로 남아도 **'같다'는 비교를 한 번 더 수행**해야 하므로, C를 찾기까지 총 4번 비교한다.

**계산 과정**

2진탐색은 정렬된 자료의 중앙값과 찾는 값을 비교하고, 찾는 값이 있을 수 있는 절반만 남겨 탐색을 반복한다. 이 문제에서는 '크다', '작다', '같다' 중 하나의 결과를 얻는 비교를 1회로 세며, **'같다'가 나올 때까지** 계속한다.

1회째: A부터 O까지 15개 자료의 중앙값은 8번째인 H이다. C는 H보다 작으므로 H와 그 오른쪽을 제외하고 A부터 G까지 7개만 남긴다.

2회째: A부터 G까지의 중앙값은 D이다. C는 D보다 작으므로 D와 그 오른쪽을 제외하고 A부터 C까지 3개만 남긴다.

3회째: A부터 C까지의 중앙값은 B이다. C는 B보다 크므로 B와 그 왼쪽을 제외한다. 이제 탐색 범위에는 C만 남지만, 아직 **'같다'는 비교 결과를 얻지 못했다**.

4회째: 남은 중앙값 C와 찾는 값 C를 비교하여 '같다'를 얻고 탐색을 종료한다.

따라서 비교 횟수는 4번이며, ④의 '4번'에 해당한다.

NoteBase

**문 8. 인터넷 서비스 관련 용어들에 대한 설명으로 가장 옳지 않은 것은?**

① ASP는 동적 맞춤형 웹페이지의 구현을 위해 사용된다.

② URL은 인터넷상에서 문서나 파일의 위치를 나타낸다.

③ HTML은 웹문서의 전달을 위한 통신 규약이다.

④ SSL은 안전한 웹통신을 위한 암호화를 위해 사용된다.

**정답 3번**

**간단한 해설**

HTML은 웹문서의 구조와 내용을 표현하는 **마크업 언어**이다. 웹문서를 요청하고 전달하는 통신 규약은 **HTTP**이다.

**선지별 해설**

① O: ASP는 웹 서버에서 스크립트를 실행하여 **동적으로 웹페이지를 생성**하는 기술이다. 사용자의 요청이나 데이터베이스 조회 결과에 따라 서로 다른 내용을 제공할 수 있어 맞춤형 웹페이지 구현에 사용된다.

② O: URL은 인터넷에서 문서나 파일 등 **자원의 위치**를 나타내는 주소이다. 예를 들어 웹페이지의 URL에는 접근에 사용할 방식, 서버의 호스트 이름, 자원의 경로 등이 포함될 수 있다.

③ X: HTML은 태그를 이용해 제목·문단·링크 등 웹문서의 구조와 내용을 표현하는 **마크업 언어**이다. 웹문서를 요청하고 전달하기 위한 통신 규약은 **HTTP**이므로, 문서의 표현 언어와 전송 규약을 구분해야 한다.

④ O: SSL은 클라이언트와 서버 사이에 오가는 데이터를 **암호화**하여 통신 내용을 보호하는 보안 프로토콜이다. 웹통신에 적용하면 제삼자가 전송 중인 데이터를 가로채더라도 내용을 쉽게 읽지 못하게 한다.

NoteBase

**문 9. <보기>의 배열 A에 n개의 원소가 있다고 가정하자. 다음 의사코드에 대한 설명으로 가장 옳지 않은 것은?**

<보기>

```
Function(A[ ], n)
{
    for last ← n downto 2   // last를 n에서 2까지 1씩 감소
        for i ← 1 to last−1
            if (A[i] > A[i+1]) then A[i] ↔ A[i+1];   // A[i]와 A[i+1]를 교환
}
```

① 제일 큰 원소를 끝자리로 옮기는 작업을 반복한다.

② 선택 정렬을 설명하는 의사코드이다.

③ $O(n^2)$의 수행 시간을 가진다.

④ 두 번째 for 루프의 역할은 가장 큰 원소를 맨 오른쪽으로 보내는 것이다.

**정답 2번**

**간단한 해설**

인접한 두 원소를 비교해 왼쪽 값이 더 크면 서로 바꾸는 **버블 정렬**이다. 선택 정렬은 정렬되지 않은 구간에서 최솟값이나 최댓값의 위치를 찾은 뒤, 그 원소를 구간의 처음이나 끝 원소와 교환한다.

NoteBase

**선지별 해설**

① O: 안쪽 반복문이 한 번 끝나면 **아직 정렬되지 않은 구간의 최댓값**이 A[last]로 이동한다. 바깥 반복문은 last를 n부터 2까지 줄이므로, 가장 큰 값부터 오른쪽 자리를 차례로 확정한다.

② X: 주어진 의사코드는 선택 정렬이 아니라 **버블 정렬**이다. A[i]와 A[i+1]이라는 인접 원소를 비교하고, 왼쪽 값이 더 클 때마다 즉시 교환한다. 선택 정렬은 구간 전체에서 최솟값이나 최댓값의 위치를 먼저 찾고, 그 원소를 구간의 경계 원소와 교환한다는 점에서 다르다.

③ O: 안쪽 반복문의 비교 횟수는 바깥 반복이 진행될 때마다 n−1회, n−2회, …, 1회가 된다. 전체 비교 횟수는 $\frac{n(n-1)}{2}$이므로 수행 시간은 $O(n^2)$이다. 이 코드에는 교환이 없을 때 종료하는 조건이 없어, **이미 정렬된 배열에서도 비교 횟수는 같다**.

④ O: 두 번째 for 루프는 i를 1부터 last−1까지 증가시키며 인접 원소를 비교한다. 각 비교에서 더 큰 값이 오른쪽에 남으므로, 끝까지 진행하면 **A[1]부터 A[last]까지의 최댓값이 A[last]에 도달**한다. 예를 들어 [3, 1, 2]를 한 차례 훑으면 [1, 3, 2]를 거쳐 [1, 2, 3]이 되어 최댓값 3이 맨 오른쪽으로 이동한다. 이후 반복에서는 이미 자리가 확정된 오른쪽 부분을 제외한 구간에 같은 작업을 한다.

NoteBase

**문 10. <보기>의 Java 프로그램의 실행 결과는?**

<보기>

```
class A {
    public void f() { System.out.print("1"); }
    public static void g() { System.out.print("2"); }
}
class B extends A {
    public void f() { System.out.print("3"); }
}
class C extends B {
    public static void g() { System.out.print("4"); }
}
public class D {
    public static void main(String args[]) {
        A obj = new C();
        obj.f();
        obj.g();
    }
}
```

① 3 2

② 3 4

③ 1 2

④ 1 4

**정답 1번**

**간단한 해설**

인스턴스 메서드 f()는 **실제 객체의 타입**에 따라 B의 구현이 실행되어 3을 출력한다. 반면 static 메서드 g()는 **참조 변수의 선언 타입 A**를 기준으로 선택되어 2를 출력한다.

NoteBase

**핵심 풀이**

객체 생성: A obj = new C();에서 참조 변수 obj의 선언 타입은 A이고, 실제 생성된 객체의 타입은 C이다. C는 B를, B는 A를 상속하므로 A 타입 변수로 C 객체를 참조할 수 있다.

obj.f(): f()는 인스턴스 메서드이므로 **실제 객체의 타입을 기준으로 재정의된 메서드**를 실행한다. C에는 f()가 없지만 B가 A의 f()를 재정의했고, C는 이를 상속한다. 따라서 B의 f()가 실행되어 3을 출력한다. A 타입 변수라는 이유로 A의 f()가 실행되는 것은 아니다.

obj.g(): g()는 static 메서드이므로 인스턴스 메서드처럼 동적으로 선택되지 않는다. C의 g()는 A의 g()를 재정의하는 것이 아니라 **숨기는 메서드**이다. obj를 통한 호출에서는 선언 타입 A를 기준으로 A의 g()가 선택되므로 2를 출력한다. 실제 객체가 C라는 이유로 C의 g()가 실행되지는 않는다.

출력 결과: obj.f()가 3을, 이어서 obj.g()가 2를 출력하므로 ①의 '3 2'에 해당한다. 다만 두 print 호출 모두 공백이나 줄바꿈을 추가하지 않으므로 실제 화면에는 두 숫자가 붙은 '32'가 출력된다.

NoteBase

**문 11. 어떤 시스템은 7비트의 데이터에 홀수 패리티 비트를 최상위 비트에 추가하여 8비트로 표현하여 저장한다. 다음과 같은 데이터를 저장 장치에서 읽어 왔을 때 오류가 발생한 경우는?**

① 011010111

② 101101111

③ 011001110

④ 101001101

**정답 1번**

**간단한 해설**

홀수 패리티에서는 패리티 비트를 포함한 전체 비트열의 **1의 개수가 홀수**여야 한다. 011010111에는 1이 6개여서 이 조건을 위반한다. 다만 문항은 8비트 저장을 설명하면서 선택지는 모두 9비트로 제시하여 길이 조건이 일치하지 않는다.

**선지별 해설**

홀수 패리티는 데이터와 패리티 비트를 합친 전체에서 1의 개수가 홀수가 되도록 하는 방식이다. 데이터에 있는 1의 개수가 짝수이면 패리티 비트를 1로, 홀수이면 0으로 정한다. 최상위 비트는 가장 왼쪽 비트이며, 패리티 검사는 **패리티 비트까지 포함한 전체**의 1을 세어 수행한다.

문항의 '7비트 데이터 + 패리티 1비트 = 8비트'라는 조건과 달리, 인쇄된 선택지는 모두 9비트이다. 따라서 주어진 저장 형식을 그대로 만족하는 예로 볼 수는 없다. 여기서는 비트를 임의로 빼거나 바꾸지 않고, **인쇄된 비트열 전체에 홀수 패리티 기준을 적용**하여 비교한다.

① 011010111: 1이 6개이다. 짝수이므로 홀수 패리티 조건을 위반하여 오류가 검출된다.

② 101101111: 1이 7개이다. 홀수이므로 홀수 패리티 검사를 통과한다.

③ 011001110: 1이 5개이다. 홀수이므로 홀수 패리티 검사를 통과한다.

④ 101001101: 1이 5개이다. 홀수이므로 홀수 패리티 검사를 통과한다.

이 기준에서 오류가 검출되는 비트열은 ①의 011010111이다. 다른 선택지의 검사 통과가 오류가 전혀 없음을 보장하는 것은 아니다.

NoteBase

**문 12. 고객, 제품, 주문, 배송업체 테이블을 가진 판매 데이터베이스를 SQL을 이용해 구축하고자 한다. 각 테이블이 <보기>와 같은 속성을 가진다고 가정할 때, 다음 중 가장 옳지 않은 SQL문은? (단, 밑줄은 기본키를 의미한다.)**

<보기>
· 고객(고객아이디, 고객이름, 나이, 등급, 직업, 적립금)
· 제품(제품번호, 제품명, 재고량, 단가, 제조업체)
· 주문(주문번호, 주문제품, 주문고객, 수량, 배송지, 주문일자)
· 배송업체(업체번호, 업체명, 주소, 전화번호)

① 고객 테이블에 가입 날짜를 추가한다. → “ALTER TABLE 고객 ADD 가입 날짜 DATE;”

② 주문 테이블에서 배송지를 삭제한다. → “ALTER TABLE 주문 DROP COLUMN 배송지;”

③ 고객 테이블에 18세 이상의 고객만 가입 가능하다는 무결성 제약 조건을 추가한다. → “ALTER TABLE 고객 ADD CONSTRAINT CHK_AGE CHECK(나이 >= 18);”

④ 배송업체 테이블을 삭제한다. → “ALTER TABLE 배송업체 DROP;”

**정답 4번**

**간단한 해설**

④의 'ALTER TABLE 배송업체 DROP;'은 테이블 자체를 삭제하는 구문이 아니다. 테이블 전체를 없애려면 **DROP TABLE 배송업체;**를 사용하며, ALTER TABLE은 기존 테이블의 열이나 제약 조건 등을 변경할 때 사용한다.

NoteBase

**선지별 해설**

① O: **열 추가 방식의 취지**는 타당하다. ALTER TABLE의 ADD 절은 기존 테이블에 새 열을 추가하며, DATE는 날짜 자료형이다. 다만 원문은 '가입 날짜'에 공백이 있는데도 열 이름을 큰따옴표로 묶지 않았으므로, 인쇄된 구문 그대로는 문법상 문제가 있다. 공백을 포함한 열 이름을 유지하려면 ALTER TABLE 고객 ADD "가입 날짜" DATE;처럼 식별자를 큰따옴표로 묶어야 한다. 열 추가 명령을 선택한 취지와 실제 식별자 표기를 구분해야 한다.

② O: ALTER TABLE 주문 DROP COLUMN 배송지;는 주문 테이블에서 **배송지 열을 삭제**한다. DROP COLUMN 다음에 삭제할 열 이름을 지정했으므로, 주문 테이블 자체를 없애는 명령과 구별된다.

③ O: ADD CONSTRAINT CHK_AGE는 CHK_AGE라는 이름의 제약 조건을 추가한다. CHECK(나이 >= 18)은 입력하거나 수정하는 나이가 18 미만이면 그 변경을 거부하도록 한다. 따라서 **나이 값이 18 이상이어야 한다는 조건**을 부여하는 방식이다. 다만 CHECK만으로는 NULL을 막지 못하므로, 나이를 반드시 입력하게 하려면 나이 열에 NOT NULL 제약도 필요하다.

④ X: ALTER TABLE 배송업체 DROP;은 삭제할 열이나 제약 조건 등도 지정하지 않은 불완전한 구문이며, 테이블 전체를 삭제하는 명령도 아니다. 배송업체 테이블 자체를 없애려면 **DROP TABLE 배송업체;**로 고쳐야 한다. ALTER TABLE 안의 DROP은 열이나 제약 조건 등을 제거하면서 테이블을 유지할 때 사용한다.

NoteBase

**문 13. <보기>의 UML 다이어그램 중 시스템의 구조(structure)보다는 주로 동작(behavior)을 묘사하는 다이어그램들만 고른 것은?**

<보기>
ㄱ. 클래스 다이어그램 (class diagram)
ㄴ. 상태 다이어그램 (state diagram)
ㄷ. 시퀀스 다이어그램 (sequence diagram)
ㄹ. 패키지 다이어그램 (package diagram)
ㅁ. 배치 다이어그램 (deployment diagram)

① ㄱ, ㄹ
② ㄴ, ㄷ
③ ㄴ, ㅁ
④ ㄷ, ㄹ

**정답 2번**

**간단한 해설**

상태 다이어그램은 **이벤트에 따른 상태 변화**를, 시퀀스 다이어그램은 **참여자 간 메시지 교환 순서**를 보여 주므로 시스템의 동작을 묘사한다.

NoteBase

**보기별 해설**

UML에서 구조 다이어그램은 시스템을 이루는 요소와 그 관계를, 동작 다이어그램은 상태 변화나 상호작용처럼 시스템이 수행하는 동작을 표현한다.

ㄱ. 클래스 다이어그램: 구조 다이어그램이다. 클래스의 속성·연산과 클래스 사이의 연관·상속 등을 표현한다. 연산을 표시하더라도 초점은 **클래스의 구성과 관계**에 있으며, 실행 중의 상태 변화나 메시지 흐름을 나타내는 것은 아니다.

ㄴ. 상태 다이어그램: 동작 다이어그램이다. 객체나 시스템이 가질 수 있는 상태와 **이벤트에 따른 상태 전이**를 표현한다. 예를 들어 주문이 결제 이벤트에 따라 결제 대기 상태에서 결제 완료 상태로 바뀌는 동작을 나타낼 수 있다.

ㄷ. 시퀀스 다이어그램: 동작 다이어그램에 속하는 상호작용 다이어그램이다. 참여자들의 생명선과 그 사이에서 주고받는 메시지를 통해 **상호작용의 진행 순서**를 표현한다. 구성 요소가 무엇인지보다 어떤 요청과 응답이 어떤 순서로 이루어지는지를 보여 준다.

ㄹ. 패키지 다이어그램: 구조 다이어그램이다. 관련 모델 요소들을 묶은 패키지와 패키지 사이의 의존 관계 등을 표현하여 **시스템의 조직 구조**를 보여 준다.

ㅁ. 배치 다이어그램: 구조 다이어그램이다. 하드웨어 장치나 실행 환경 등의 노드에 소프트웨어 산출물이 어떻게 배치되는지를 표현한다. 실행 환경을 다루지만, 초점은 동작의 흐름이 아니라 **노드와 소프트웨어 산출물의 배치 관계**에 있다.

동작을 주로 묘사하는 것은 ㄴ과 ㄷ이며, 이 조합은 ②에 해당한다.

NoteBase

**문 14. <보기 1>의 테이블 R에 대해 <보기 2>의 SQL을 수행한 결과로 옳은 것은?**

<보기 1>

| A | B |
|---|---|
| 3 | 1 |
| 2 | 4 |
| 3 | 2 |
| 2 | 5 |
| 3 | 3 |
| 1 | 5 |

<보기 2>

```
SELECT SUM(B) FROM R GROUP BY A
HAVING COUNT(B) = 2;
```

① 2

② 5

③ 6

④ 9

**정답 4번**

**간단한 해설**

A별로 묶은 뒤 **COUNT(B) = 2인 그룹**만 남긴다. A가 2인 그룹의 B값은 4와 5이므로 합계는 **9**이다.

NoteBase

**계산 과정**

GROUP BY A는 **A값이 같은 행끼리** 묶는다. A가 1인 그룹의 B값은 5, A가 2인 그룹의 B값은 4·5, A가 3인 그룹의 B값은 1·2·3이다.

HAVING은 그룹에 조건을 적용한다. COUNT(B)는 각 그룹에서 NULL이 아닌 B값의 개수를 세며, 이 테이블에는 B의 NULL값이 없다. A가 1·2·3인 그룹의 개수는 각각 1·2·3이므로 **A가 2인 그룹만** COUNT(B) = 2를 만족한다.

SELECT SUM(B)는 조건을 통과한 그룹의 B값을 더한다. 남은 그룹의 B값은 4와 5이므로 4 + 5 = 9이다. SQL의 실행 결과는 9 한 개이며, ④의 값과 일치한다.

NoteBase

**문 15. <보기>는 데이터가 정렬되는 단계를 일부 보여준 것이다. 어떤 정렬 알고리즘을 사용하면 이와 같은 데이터의 자리 교환이 일어나겠는가? (단, 제일 위의 행이 주어진 데이터이고, 아래로 내려갈수록 정렬이 진행되는 것이다.)**

<보기>

| | |
|---|---|
| 초기 데이터 | 8 9 4 3 7 1 5 2 |
| | 8 9 3 4 1 7 2 5 |
| | 3 4 8 9 1 2 5 7 |
| 정렬 데이터 | 1 2 3 4 5 7 8 9 |

① 삽입 정렬

② 선택 정렬

③ 합병 정렬

④ 퀵 정렬

**정답 3번**

**간단한 해설**

각 단계에서 정렬된 부분 배열의 크기가 **2개 → 4개 → 8개**로 커진다. 작은 정렬 배열 두 개를 하나의 큰 정렬 배열로 합치는 **합병 정렬**의 과정이다.

NoteBase

**핵심 풀이**

합병 정렬은 이미 정렬된 두 부분 배열에서 앞쪽 값을 비교하여 작은 값부터 차례로 가져오는 방식으로, 하나의 정렬된 배열을 만든다. 보기에서는 아래로 내려갈수록 **정렬된 부분 배열끼리 합쳐지는** 모습을 확인할 수 있다.

두 원소씩 정렬: 초기 데이터 8, 9, 4, 3, 7, 1, 5, 2를 인접한 두 원소씩 처리하면 [8, 9], [3, 4], [1, 7], [2, 5]가 된다. 이것이 두 번째 행의 8, 9, 3, 4, 1, 7, 2, 5이다.

네 원소씩 합병: [8, 9]와 [3, 4]를 합치면 [3, 4, 8, 9]가 되고, [1, 7]과 [2, 5]를 합치면 [1, 2, 5, 7]이 된다. 예를 들어 뒤쪽 두 배열에서는 앞의 값 1과 2를 비교해 1을 가져오고, 이어 7과 2를 비교해 2를 가져오는 식으로 정렬 순서를 만든다. 두 결과를 이어 놓으면 세 번째 행과 같다.

전체 합병: 정렬된 [3, 4, 8, 9]와 [1, 2, 5, 7]을 다시 합치면 마지막 행의 [1, 2, 3, 4, 5, 7, 8, 9]가 된다. 이처럼 정렬된 부분 배열의 크기를 두 배씩 늘려 전체를 정렬하는 과정이 합병 정렬에 해당한다.

퀵 정렬과의 구별: 퀵 정렬도 분할 정복을 이용하지만, **피벗을 기준으로 작은 값과 큰 값을 나누는** 것이 핵심이다. 보기의 결정적 특징은 피벗에 따른 분할이 아니라 정렬된 두 부분 배열의 합병이다.

NoteBase

**문 16. <보기>의 각 설명과 일치하는 데이터 구조로 바르게 짝지어진 것은?**

<보기>
(가) 먼저 추가된 항목이 먼저 제거된다.
(나) 먼저 추가된 항목이 나중에 제거된다.
(다) 항목이 추가된 순서에 상관없이 제거된다.

① (가): 큐, (나): 연결 리스트, (다): 스택
② (가): 스택, (나): 연결 리스트, (다): 큐
③ (가): 스택, (나): 큐, (다): 연결 리스트
④ (가): 큐, (나): 스택, (다): 연결 리스트

**정답 4번**

**간단한 해설**

큐는 **먼저 넣은 항목을 먼저** 꺼내고, 스택은 **나중에 넣은 항목을 먼저** 꺼낸다. 연결 리스트는 삽입 시점에 따른 삭제 순서가 정해져 있지 않아 원하는 항목을 찾아 삭제할 수 있다.

**대응 관계 해설**

(가) 큐: 먼저 추가된 항목을 먼저 제거하는 **선입선출(FIFO)** 구조이다. 뒤쪽에서 항목을 넣고 앞쪽에서 꺼내므로, A를 넣고 B를 넣으면 A가 먼저 제거된다.

(나) 스택: 나중에 추가된 항목을 먼저 제거하는 **후입선출(LIFO)** 구조이다. 같은 쪽 끝에서 항목을 넣고 꺼내므로, A를 넣고 B를 넣으면 B가 먼저 제거되고 먼저 넣은 A는 나중에 제거된다.

(다) 연결 리스트: 데이터와 다음 노드를 가리키는 링크 등을 가진 노드들을 연결한 구조이다. **삭제할 노드를 찾아 연결을 바꾸면** 해당 항목을 제거할 수 있으므로, 먼저 넣었는지 나중에 넣었는지에 따라 삭제 순서가 강제되지 않는다. '순서에 상관없이'는 무작위로 삭제한다는 뜻이 아니라, 큐나 스택과 같은 삽입 순서 기반의 삭제 규칙이 없다는 뜻이다.

따라서 (가)는 큐, (나)는 스택, (다)는 연결 리스트로 대응하며, 이 조합은 ④와 일치한다.

NoteBase

**문 17. 전화번호의 마지막 네 자리를 3으로 나눈 나머지를 해싱(hashing) 하여 데이터베이스에 저장하고자 한다. 나머지 셋과 다른 저장 장소에 저장되는 것은?**

① 010-4021-6718

② 010-9615-4815

③ 010-7290-6027

④ 010-2851-5232

**정답 1번**

**간단한 해설**

6718을 3으로 나누면 **나머지가 1**이고, 다른 세 전화번호의 마지막 네 자리는 모두 3의 배수이므로 나머지가 0이다. 저장 장소는 이 나머지로 결정된다.

**계산 과정**

이 문제에서는 전화번호의 마지막 네 자리를 3으로 나눈 나머지가 저장 장소를 결정한다. 나머지는 0, 1, 2 중 하나이며, **나머지가 같으면 같은 저장 장소**에 대응한다.

① 010-4021-6718: 6718 = 3 × 2239 + 1이므로 나머지는 1이다.

② 010-9615-4815: 4815 = 3 × 1605이므로 나머지는 0이다.

③ 010-7290-6027: 6027 = 3 × 2009이므로 나머지는 0이다.

④ 010-2851-5232: 5232 = 3 × 1744이므로 나머지는 0이다.

②·③·④는 나머지가 모두 0이어서 같은 저장 장소에 대응하고, ①은 **나머지가 1로 달라** 나머지 셋과 다른 저장 장소에 저장된다.

NoteBase

**문 18. 다음 메모리 영역 중 전역 변수가 저장되는 영역은?**

① 데이터(Data)

② 스택(Stack)

③ 텍스트(Text)

④ 힙(Heap)

**정답 1번**

**간단한 해설**

전역 변수는 프로그램 실행 동안 저장 공간이 유지되며, 일반적인 메모리 영역 구분에서 **데이터 영역**에 저장된다.

**공통 해설**

일반적인 프로그램 메모리 구조에서 데이터(Data) 영역은 **전역 변수와 정적 변수**의 저장 공간이다. 이 변수들은 특정 함수의 실행이 끝나더라도 저장 공간이 사라지지 않고 프로그램 실행이 끝날 때까지 유지된다. 예를 들어 여러 함수가 같은 전역 변수를 읽고 값을 바꾸더라도 그 변수의 저장 공간은 계속 유지된다. 따라서 전역 변수의 저장 영역은 데이터 영역이다.

NoteBase

**문 19. UML(Unified Modeling Language)에 대한 설명으로 가장 옳지 않은 것은?**

① UML은 방법론으로, 단계별로 어떻게 작업해야 하는지 자세하게 나타낸다.

② UML은 소프트웨어의 구성요소와 그것들의 관계 및 상호작용을 시각화한 것이다.

③ UML은 객체지향 소프트웨어를 모델링하는 표준 그래픽 언어로, 심벌과 그림을 사용해 객체지향 개념을 나타낼 수 있다.

④ UML은 소프트웨어 개발의 중요한 작업인 분석, 설계, 구현의 정확하고 완벽한 모델을 제공한다.

**정답 1번**

**간단한 해설**

UML은 단계별 작업 절차를 정하는 방법론이 아니라, 소프트웨어의 구조와 동작을 표현하는 **표준 모델링 언어**이다. 개발 방법론이 '어떤 순서와 방법으로 작업할지'를 정한다면, UML은 그 결과를 '어떤 기호와 규칙으로 표현할지'를 정한다.

**선지별 해설**

① X: UML은 **개발 절차를 규정하는 방법론이 아니다**. 방법론은 요구사항 분석이나 설계 등을 어떤 절차와 방법으로 수행할지 정하고, UML은 그 과정에서 얻은 결과를 표현하는 언어이다. 따라서 '방법론으로, 단계별로 어떻게 작업해야 하는지 자세하게 나타낸다'는 부분은 '모델링 언어로, 소프트웨어의 구조와 동작을 표현한다'로 고쳐야 한다.

② O: UML은 소프트웨어를 이루는 요소뿐 아니라 요소 사이의 **관계와 상호작용**도 시각적으로 표현한다. 예를 들어 클래스 다이어그램은 클래스의 구조와 관계를, 시퀀스 다이어그램은 객체 사이에 메시지가 오가는 순서를 나타낸다.

③ O: UML은 객체지향 개념을 표현하는 **표준 그래픽 모델링 언어**이다. 클래스·객체·연관·상속 등을 정해진 기호와 다이어그램으로 나타내므로, 개발자들이 같은 표기법으로 시스템을 이해하고 의사소통할 수 있다.

④ O: UML은 분석·설계·구현에서 필요한 내용을 **명확하고 충분하게 명세**하는 데 쓰인다. 분석에서는 요구 기능을, 설계에서는 클래스와 상호작용을, 구현에서는 구성요소와 배치 구조 등을 모델로 표현할 수 있다. 선지의 '정확하고 완벽한 모델'은 이러한 명세 기능을 가리키며, UML을 사용하기만 하면 모델의 누락이나 구현 오류가 자동으로 없어진다는 뜻은 아니다.

NoteBase

**문 20. <보기>의 C 프로그램을 실행했을 때, 화면에 출력되는 값은? (단, 프로그램의 첫 번째 열의 숫자는 행 번호이고 프로그램의 일부는 아님.)**

<보기>

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #define N 3
4  int main(void){
5      int (*in)[N], *out, sum=0;
6
7      in=(int (*)[N])malloc(N*N*sizeof(int));
8      out=(int *)in;
9
10     for (int i=0; i<N*N; i++)
11         out[i]=i;
12
13     for (int i=0; i<N; i++)
14         sum += in[i][i];
15
16     printf("%d", sum);
17     return 0;
18 }
```

① 0

② 3

③ 6

④ 12

**정답 4번**

**간단한 해설**

in은 정수 3개를 한 행으로 접근하므로, in[i][i]가 읽는 값은 **대각 원소 0, 4, 8**이다. 이들을 더한 12가 출력된다.

NoteBase

**계산 과정**

5~8행: N은 3이며, in은 정수 3개로 이루어진 배열을 가리키는 포인터이다. malloc으로 정수 3 × 3 = 9개를 저장할 연속 공간을 확보한다. out은 같은 공간의 시작을 가리키는 int형 포인터이므로, out으로는 정수를 하나씩, in으로는 **정수 3개를 한 행으로** 접근한다. sum의 초깃값은 0이다.

10~11행: i가 0부터 8까지 변하면서 out[i] = i를 실행한다. 따라서 저장된 값은 차례로 0, 1, 2, 3, 4, 5, 6, 7, 8이다. 같은 공간을 in으로 읽으면 0번 행은 0·1·2, 1번 행은 3·4·5, 2번 행은 6·7·8이 된다.

13~14행: i는 0, 1, 2를 거치며, in[i][i]는 **행 번호와 열 번호가 같은 원소**를 읽는다. in[0][0] = 0, in[1][1] = 4, in[2][2] = 8이므로, sum은 0 → 0 → 4 → 12로 변한다.

16행: %d는 sum을 십진 정수로 출력한다. 최종 합은 0 + 4 + 8 = 12이므로 출력값은 12이며, ④의 값과 일치한다.

NoteBase