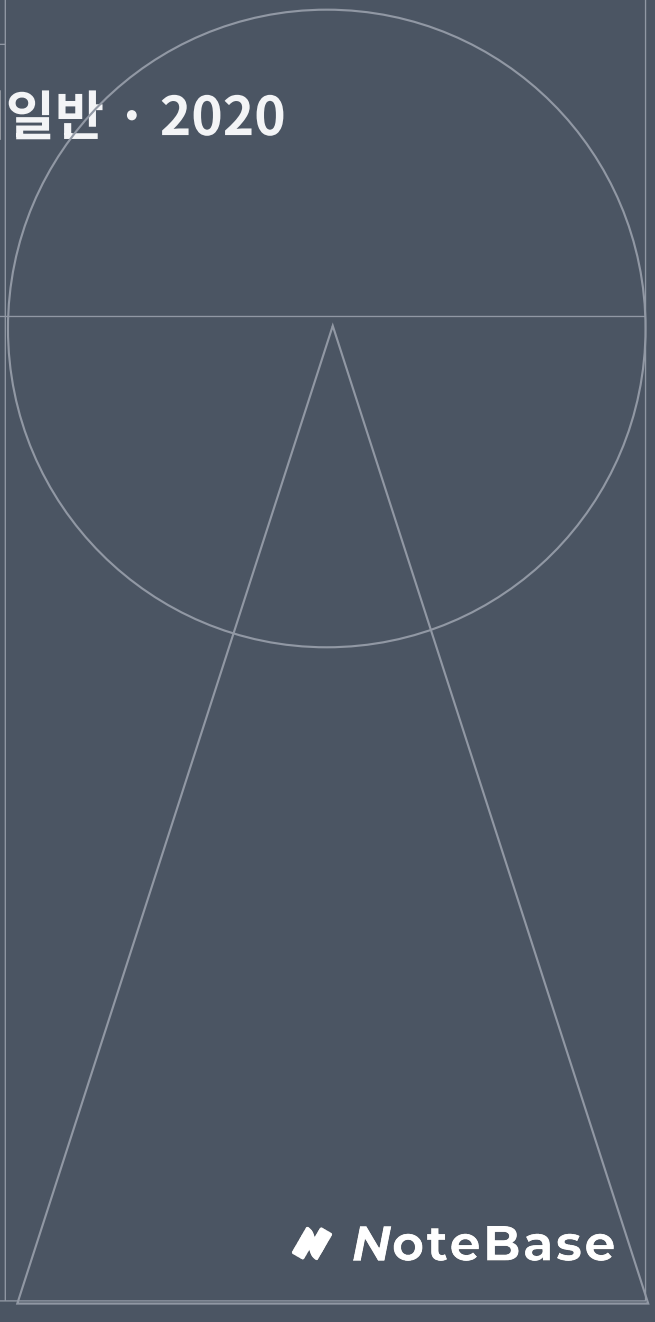
기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2020



NoteBase

기출OX 해설집

공무원/국가직/9급 · 컴퓨터일반 · 2020

공무원/국가직/9급 · 컴퓨터일반 · 2020

문 1. 아날로그 신호를 디지털 신호로 변조하기 위한 펄스부호변조(PCM) 과정으로 옳지 않은 것은?

① 분절화(Segmentation)

② 표본화(Sampling)

③ 부호화(Encoding)

④ 양자화(Quantization)

정답 1번

간단한 해설

PCM의 기본 변환 과정은 **표본화·양자화·부호화**이며, **분절화는 포함되지 않는다**.

개념 구분

PCM은 아날로그 신호를 표본화하고, 얻은 표본값을 양자화한 뒤 부호화하여 디지털로 표현하는 방식이다.

표본화(Sampling): 시간에 따라 연속적으로 변하는 신호에서 **일정한 시간 간격으로 신호값을 추출**한다. 신호를 읽는 시점이 이산적으로 정해지지만, 추출한 값의 크기는 아직 연속적인 값을 가질 수 있다.

양자화(Quantization): 추출한 표본값을 **미리 정한 유한한 단계의 대표값**에 대응시킨다. 표본화가 시간축을 이산화한다면, 양자화는 신호값의 크기를 이산화한다.

부호화(Encoding): 양자화한 각 대표값을 **0과 1로 이루어진 이진 부호**로 표현한다. 이 과정을 거치면 각 표본을 비트열로 나타낼 수 있다.

분절화(Segmentation): PCM의 기본 변환 단계에 해당하지 않는다. 신호를 일정한 시간 간격으로 읽는 과정과 혼동할 수 있으나, 그 과정의 명칭은 **표본화**이다.

NoteBase

문 2. DBMS를 사용하는 이점으로 옳지 않은 것은?

① 데이터를 프로그램과 분리함으로써 데이터 독립성이 향상된다.

② 데이터의 공유와 동시 접근이 가능하다.

③ 데이터의 중복을 허용하여 데이터의 일관성을 유지한다.

④ 데이터의 무결성과 보안성을 유지한다.

정답 3번

간단한 해설

DBMS는 데이터의 중복 자체로 일관성을 유지하는 것이 아니라, **불필요한 중복을 줄이고 중복된 값이 서로 일치하도록 관리**한다. 같은 데이터를 여러 곳에 저장하면 일부만 수정되어 값이 달라질 수 있다.

선지별 해설

① O: 데이터 독립성은 데이터의 구조나 저장 방식이 바뀌어도 응용 프로그램이 받는 영향을 줄이는 성질이다. DBMS가 프로그램과 데이터 사이에서 접근을 관리하므로, 프로그램이 **물리적인 저장 방식에 직접 의존하는 정도**를 줄일 수 있다.

② O: DBMS는 여러 사용자와 프로그램이 같은 데이터를 공유하고 동시에 접근할 수 있게 한다. 이때 **동시성 제어**를 통해 여러 작업이 서로 간섭하여 데이터가 잘못 처리되는 것을 막는다.

③ X: 같은 데이터를 여러 곳에 중복 저장하면 일부만 갱신되어 서로 다른 값이 남을 수 있다. 예를 들어 같은 사람의 전화번호를 두 곳에 저장하고 한 곳만 수정하면 어느 번호가 현재 값인지 불분명해진다. 따라서 '중복을 허용하여'는 **'불필요한 중복을 줄이고 필요한 중복을 통제하여'**로 고쳐야 한다. 일관성은 같은 사실을 나타내는 데이터가 서로 모순되지 않도록 관리할 때 유지된다.

④ O: DBMS는 무결성 제약조건으로 데이터가 정해진 규칙을 만족하도록 하고, 사용자 인증과 접근 권한 관리로 허가되지 않은 접근을 제한한다. 예를 들어 기본키 값의 중복을 막는 것은 **무결성 유지**이고, 특정 사용자에게만 수정 권한을 주는 것은 **보안 관리**이다.

NoteBase

문 3. CPU 내의 레지스터에 대한 설명으로 옳지 않은 것은?

① Accumulator(AC): 연산 과정의 데이터를 일시적으로 저장하는 레지스터

② Program Counter(PC): 다음에 인출될 명령어의 주소를 보관하는 레지스터

③ Memory Address Register(MAR): 가장 최근에 인출한 명령어를 보관하는 레지스터

④ Memory Buffer Register(MBR): 기억장치에 저장될 데이터 혹은 기억장치로부터 읽힌 데이터가 일시적으로 저장되는 버퍼 레지스터

정답 3번

간단한 해설

MAR은 기억장치에서 읽거나 쓸 위치의 **주소**를 보관한다. 가장 최근에 인출한 **명령어 자체는 명령어 레지스터(IR)**에 보관한다.

선지별 해설

① O: 누산기(AC)는 연산에 사용할 데이터나 연산의 중간 결과를 일시적으로 저장한다. 연산 결과를 다시 누산기에 두면 **다음 연산의 입력**으로 사용할 수 있다.

② O: 프로그램 카운터(PC)는 **다음에 인출할 명령어의 주소**를 보관한다. 명령어 자체를 저장하는 것이 아니라, 다음 명령어를 기억장치의 어디에서 가져올지 나타낸다.

③ X: 메모리 주소 레지스터(MAR)는 기억장치에서 **읽거나 쓸 위치의 주소**를 보관한다. 가장 최근에 인출한 명령어를 보관하는 것은 명령어 레지스터(IR)이다. 명령어를 인출할 때 MAR은 그 명령어가 있는 위치를 지정하고, IR은 인출된 명령어 자체를 보관하여 해독에 사용한다.

④ O: 메모리 버퍼 레지스터(MBR)는 CPU와 기억장치 사이에서 **전달되는 데이터**를 일시적으로 보관한다. 기억장치에 쓸 때는 저장할 데이터를, 기억장치에서 읽을 때는 읽어 온 데이터를 담는다. 따라서 접근할 주소를 담는 MAR과 역할이 다르다.

NoteBase

문 4. 소프트웨어 개발 프로세스 중 원형(Prototyping) 모델의 단계별 진행 과정을 올바르게 나열한 것은?

① 요구 사항 분석 → 시제품 설계 → 고객의 시제품 평가 → 시제품 개발 → 시제품 정제 → 완제품 생산
② 요구 사항 분석 → 시제품 설계 → 시제품 개발 → 고객의 시제품 평가 → 시제품 정제 → 완제품 생산
③ 요구 사항 분석 → 고객의 시제품 평가 → 시제품 개발 → 시제품 설계 → 시제품 정제 → 완제품 생산
④ 요구 사항 분석 → 시제품 개발 → 시제품 설계 → 고객의 시제품 평가 → 시제품 정제 → 완제품 생산

정답 2번

간단한 해설

원형 모델은 시제품을 만들어 고객의 요구를 확인하고 보완하는 방식이다. **설계한 뒤 개발**해야 평가할 시제품이 생기며, **고객의 평가를 반영**하여 시제품을 정제한다.

올바른 순서

요구 사항 분석 → 시제품 설계 → 시제품 개발 → 고객의 시제품 평가 → 시제품 정제 → 완제품 생산

요구 사항 분석: 고객이 원하는 기능과 해결할 문제를 파악한다. 이때 확인한 요구 사항이 시제품에서 무엇을 보여줄지 정하는 기준이 된다.

시제품 설계: 파악한 요구 사항을 바탕으로 주요 기능과 화면 등의 구성을 정한다. 실제로 만들 대상을 구체화하는 단계이므로 **시제품 개발보다 앞선다**.

시제품 개발: 설계한 내용을 바탕으로 고객이 확인할 수 있는 시제품을 만든다. 고객이 기능과 사용 방식을 평가하려면 먼저 **평가 대상인 시제품**이 있어야 한다.

고객의 시제품 평가: 고객이 시제품을 확인하여 요구에 맞는 부분과 부족한 부분을 판단한다. 이 평가에서 얻은 의견이 다음 단계의 수정·보완 근거가 된다.

시제품 정제: **고객의 평가 결과를 반영**하여 기능과 구성을 수정·보완한다. 요구 사항이 충분히 반영될 때까지 시제품을 개선하고 다시 평가받는 과정을 반복할 수 있다.

완제품 생산: 시제품을 통해 확인하고 구체화한 요구 사항을 바탕으로 최종 소프트웨어를 완성한다. 고객의 요구를 확인하고 필요한 보완을 거친 뒤 최종 제품을 만드는 단계이다.

NoteBase

문 5. 네트워크 토폴로지에 대한 설명으로 옳지 않은 것은?

① 버스(bus)형 토폴로지는 설치가 간단하고 비용이 저렴하다.

② 링(ring)형 토폴로지는 통신 회선에 컴퓨터를 추가하거나 삭제하는 등 네트워크 재구성이 용이하다.

③ 트리(tree)형 토폴로지는 허브(hub)에 문제가 발생해도 전체 네트워크에 영향을 주지 않는다.

④ 성(star)형 토폴로지는 중앙집중적인 구조이므로 고장 발견과 유지보수가 쉽다.

정답 3번

간단한 해설

트리형은 허브를 계층적으로 연결하므로 **상위 허브의 장애가 하위 네트워크로 영향을 미친다**. 특히 최상위 허브가 고장 나면 전체 네트워크의 통신이 마비될 수 있다.

선지별 해설

① O: 버스형은 하나의 공통 통신 회선에 여러 컴퓨터를 연결하는 구조이다. 중앙 허브나 컴퓨터마다 별도의 전용 회선을 둘 필요가 없어 **설치가 간단하고 구축 비용이 저렴하다**.

② O: 링형은 각 컴퓨터를 양옆의 두 컴퓨터와 연결하여 고리 모양을 이룬다. 컴퓨터를 추가하거나 삭제할 때 **해당 위치의 인접 연결을 변경**하면 되므로 연결 구조의 재구성이 비교적 간단하다. 다만 재구성이 쉽다는 것이 통신 중단 없이 변경할 수 있다는 뜻은 아니며, 단일 링에서는 연결을 바꾸는 동안 통신이 영향을 받을 수 있다.

③ X: 트리형은 상위 허브 아래에 하위 허브와 컴퓨터를 가지처럼 연결하는 계층적 구조이다. 상위 허브를 거쳐야 다른 가지와 통신할 수 있으므로, 허브 장애는 **그 허브에 의존하는 하위 네트워크의 통신에 영향**을 준다. 특히 최상위 허브가 고장 나면 전체 네트워크의 통신이 마비될 수 있으므로, '전체 네트워크에 영향을 주지 않는다'는 설명은 잘못이다.

④ O: 성형은 각 컴퓨터를 중앙 허브에 개별 회선으로 연결하는 구조이다. 중앙에서 연결 상태를 확인하고 문제가 생긴 회선을 따로 점검할 수 있어 **고장 발견과 유지보수가 쉽다**. 개별 컴퓨터나 그 연결 회선의 장애는 다른 컴퓨터에 직접 영향을 주지 않지만, 중앙 허브가 고장 나면 전체 네트워크의 통신이 중단될 수 있다.

NoteBase

문 6. RAID(Redundant Array of Independent Disks) 레벨에 대한 설명으로 옳지 않은 것은?

① RAID 1 구조는 데이터를 두 개 이상의 디스크에 패리티 없이 중복 저장한다.

② RAID 2 구조는 데이터를 각 디스크에 비트 단위로 분산 저장하고 여러 개의 해밍코드 검사디스크를 사용한다.

③ RAID 4 구조는 각 디스크에 데이터를 블록 단위로 분산 저장하고 하나의 패리티 검사디스크를 사용한다.

④ RAID 5 구조는 각 디스크에 데이터와 함께 이중 분산 패리티 정보를 블록 단위로 분산 저장한다.

정답 4번

간단한 해설

④의 '이중 분산 패리티'는 **RAID 6**의 특징이다. RAID 5는 데이터를 블록 단위로 나누어 저장하고 **단일 패리티를 여러 디스크에 분산**한다.

선지별 해설

① O: RAID 1은 같은 데이터를 두 개 이상의 디스크에 중복 저장하는 **미러링** 방식이다. 패리티를 계산하여 복구하는 대신 데이터의 복사본을 유지하므로, 한 디스크에 장애가 발생하면 다른 디스크의 복사본을 사용할 수 있다.

② O: RAID 2는 데이터를 **비트 단위로 분산 저장**하고, 여러 검사 디스크에 해밍코드의 검사 정보를 저장한다. 해밍코드는 오류 위치를 식별하고 오류를 정정하는 데 사용하는 오류 정정 코드이다.

③ O: RAID 4는 데이터를 블록 단위로 여러 디스크에 나누어 저장하고, **하나의 전용 패리티 디스크**에 복구용 정보를 저장한다. 한 데이터 디스크가 고장 나면 나머지 데이터와 패리티를 이용해 잃어버린 데이터를 복구할 수 있다.

④ X: RAID 5는 블록 단위로 데이터를 분산 저장하면서 각 스트라이프의 **단일 패리티**도 여러 디스크에 분산한다. 스트라이프는 여러 디스크의 대응하는 블록들을 묶은 단위이다. 패리티를 한 전용 디스크에 모으는 RAID 4와 달리, 스트라이프마다 패리티를 맡는 디스크를 달리한다. 선지의 '이중 분산 패리티'는 **RAID 6**의 특징이다. RAID 5는 한 디스크의 장애를 견디며, RAID 6는 서로 독립적인 두 패리티 정보를 사용하여 두 디스크의 동시 장애까지 견딘다.

NoteBase

문 7. 다중 스레드(Multi Thread) 프로그래밍의 이점에 대한 설명으로 옳지 않은 것은?

① 다중 스레드는 사용자의 응답성을 증가시킨다.

② 스레드는 그들이 속한 프로세스의 자원들과 메모리를 공유한다.

③ 프로세스를 생성하는 것보다 스레드를 생성하여 문맥을 교환하면 오버헤드가 줄어든다.

④ 다중 스레드는 한 스레드에 문제가 생기더라도 전체 프로세스에 영향을 미치지 않는다.

정답 4번

간단한 해설

같은 프로세스의 스레드들은 **메모리와 자원을 공유**한다. 따라서 한 스레드가 공유 데이터를 훼손하거나 치명적 오류를 일으키면 **전체 프로세스에 영향을 줄 수 있다**.

선지별 해설

① O: 작업을 여러 스레드로 나누면 한 스레드가 오래 걸리는 작업이나 입출력을 처리하는 동안 다른 스레드가 사용자 입력에 응답할 수 있다. 예를 들어 파일을 읽는 작업과 화면 입력 처리를 나누면 파일 읽기가 끝날 때까지 화면 전체가 멈추는 현상을 줄여 **응답성을 높일 수 있다**.

② O: 같은 프로세스에 속한 스레드들은 **코드·데이터·힙과 열린 파일 등의 자원을 공유**한다. 다만 각 스레드는 실행 위치와 상태를 따로 유지해야 하므로 프로그램 카운터, 레지스터, 스택은 각각 가진다.

③ O: 스레드는 기존 프로세스의 메모리와 자원을 활용하므로 별도의 프로세스를 만드는 것보다 일반적으로 생성 비용이 작다. 또한 **같은 프로세스 안의 스레드 간 문맥 교환**은 주소 공간을 바꿀 필요가 없어 프로세스 간 문맥 교환보다 일반적으로 오버헤드가 작다. 문맥 교환은 실행 중인 작업의 상태를 저장하고 다음 작업의 상태를 복원하는 과정이다.

④ X: 같은 프로세스의 스레드들은 주소 공간을 공유하므로 **오류의 영향이 다른 스레드로 번질 수 있다**. 한 스레드가 공유 데이터를 잘못 변경하면 다른 스레드도 잘못된 데이터를 사용하게 되며, 치명적 오류로 프로세스가 종료되면 그 안의 모든 스레드도 함께 종료된다. 따라서 '전체 프로세스에 영향을 미치지 않는다'는 '전체 프로세스에 영향을 줄 수 있다'로 고쳐야 한다.

NoteBase

문 8. OSI(Open Systems Interconnect) 모델에 대한 설명으로 옳지 않은 것은?

① 네트워크 계층은 데이터 전송에 관한 서비스를 제공하는 계층으로 송신 측과 수신 측 사이의 실제적인 연결 설정 및 유지, 오류 복구와 흐름 제어 등을 수행한다.

② 데이터링크 계층은 네트워크 계층에서 받은 데이터를 프레임(frame)이라는 논리적인 단위로 구성하고 전송에 필요한 정보를 덧붙여 물리 계층으로 전달한다.

③ 세션 계층은 전송하는 두 종단 프로세스 간의 접속(session)을 설정하고, 유지하고 종료하는 역할을 한다.

④ 표현 계층은 전송하는 데이터의 표현 방식을 관리하고 암호화하거나 데이터를 압축하는 역할을 한다.

정답 1번

간단한 해설

송수신 종단 사이의 연결 관리와 신뢰성 있는 전송을 위한 오류 복구·흐름 제어는 **전송 계층**의 역할이다.
네트워크 계층은 **논리 주소와 경로 선택**을 바탕으로 패킷을 목적지까지 전달한다.

선지별 해설

① X: 송신 측과 수신 측 사이의 종단 간 연결을 관리하고, 신뢰성 있는 전송을 위해 오류 복구와 흐름 제어를 수행하는 것은 **전송 계층**의 역할이다. 네트워크 계층은 논리 주소를 이용해 목적지를 식별하고 경로를 선택하여, 여러 네트워크를 거쳐 패킷을 전달하는 역할을 한다. 따라서 선지의 계층명을 '전송 계층'으로 고쳐야 한다.

② O: 데이터링크 계층은 상위의 네트워크 계층에서 받은 패킷에 제어 정보를 덧붙여 **프레임 단위**로 구성한다. 프레임에는 같은 링크에서의 전달과 오류 검출 등에 필요한 정보가 포함되며, 이를 물리 계층에 넘겨 비트 신호로 전송하게 한다.

③ O: 세션 계층은 두 종단 프로세스 사이에서 대화를 진행하기 위한 **세션의 설정·유지·종료**를 관리한다. 여기서 세션은 통신하는 프로세스 사이의 논리적인 대화 관계이며, 물리적인 회선 연결 자체를 뜻하지 않는다.

④ O: 표현 계층은 서로 다른 시스템이 데이터를 같은 의미로 해석할 수 있도록 **데이터의 표현 형식**을 다룬다. 문자 인코딩 등의 형식을 변환하고, 데이터의 기밀성을 위한 암호화와 전송량을 줄이기 위한 압축도 수행한다.

NoteBase

문 9. 캐시기억장치 교체 알고리즘에 대한 설명으로 옳지 않은 것은?

① LRU는 최근에 가장 오랫동안 사용되지 않았던 블록을 교체하는 방법이다.

② FIFO는 캐시에 적재된 지 가장 오래된 블록을 먼저 교체하는 방법이다.

③ LFU는 캐시 블록마다 참조 횟수를 기록함으로써 가장 많이 참조된 블록을 교체하는 방법이다.

④ Random은 사용 횟수와 무관하게 임의로 블록을 교체하는 방법이다.

정답 3번

간단한 해설

LFU는 참조 횟수가 **가장 적은 블록**을 교체한다. ③의 '가장 많이 참조된 블록'은 '가장 적게 참조된 블록'으로 고쳐야 한다.

선지별 해설

① O: LRU(Least Recently Used)는 **마지막 참조 이후 가장 오랜 시간이 지난 블록**을 교체한다. 예를 들어 A와 B를 차례로 적재한 뒤 A를 다시 참조했다면, 두 블록 중에서는 B가 더 오랫동안 사용되지 않았으므로 교체 대상이 된다.

② O: FIFO(First In First Out)는 **캐시에 들어온 순서**를 기준으로 가장 먼저 적재된 블록을 교체한다. A와 B를 차례로 적재한 뒤 A를 다시 참조하더라도, 적재 순서는 바뀌지 않으므로 두 블록 중에서는 A가 교체 대상이다. 최근 참조 시점을 따지는 LRU와 구별된다.

③ X: LFU(Least Frequently Used)는 블록별 참조 횟수를 기록하여 **참조 횟수가 가장 적은 블록**을 교체한다. 적게 참조된 블록보다 자주 참조된 블록을 캐시에 남기는 방식이므로, '가장 많이 참조된 블록을 교체'한다는 설명은 반대이다.

④ O: Random은 교체 대상 블록을 **무작위로 선택**한다. 참조 횟수나 최근 참조 시점, 적재 순서를 교체 기준으로 삼지 않는다.

NoteBase

문 10. 8진수 123.321을 16진수로 변환한 것은?

① 53.35

② 53.321

③ 53.681

④ 53.688

정답 4번

간단한 해설

8진수 한 자리를 2진수 3자리로 바꾼 뒤, **소수점을 기준으로 정수부는 왼쪽으로, 소수부는 오른쪽으로 4자리씩** 묶는다. 부족한 자리를 0으로 채우면 16진수 $(53.688)_{16}$이 된다.

계산 과정

8진수 $(123.321)_8$과 값이 같은 16진수를 구한다. $8 = 2^3$, $16 = 2^4$이므로 8진수 한 자리는 2진수 3자리, 16진수 한 자리는 2진수 4자리에 대응한다. 따라서 2진수를 거쳐 자릿수를 다시 묶으면 된다.

정수부 123: 각 자리를 3비트로 바꾸면 1 → 001, 2 → 010, 3 → 011이므로 001 010 011이다. 이를 **소수점에서 왼쪽으로 4비트씩** 묶고 맨 앞의 불필요한 0을 정리하면 0101 0011이 된다. 0101은 5, 0011은 3이므로 정수부는 16진수 53이다.

소수부 .321: 각 자리를 3비트로 바꾸면 3 → 011, 2 → 010, 1 → 001이므로 .011 010 001이다. 이번에는 **소수점에서 오른쪽으로 4비트씩** 묶어 .0110 1000 1로 나눈다. 마지막 묶음의 오른쪽에 0 세 개를 붙이면 .0110 1000 1000이 된다. 소수부 끝에 0을 붙여도 값은 변하지 않는다. 각 묶음은 차례로 6, 8, 8이므로 소수부는 16진수 .688이다.

정수부와 소수부를 합치면 $(123.321)_8 = (53.688)_{16}$이며, 이는 ④의 53.688과 일치한다.

NoteBase

문 11. 암호화 기술에 대한 설명으로 옳은 것은?

① 공개키 암호화는 암호화하거나 복호화하는 데 동일한 키를 사용한다.

② 공개키 암호화는 비공개키 암호화에 비해 암호화 알고리즘이 복잡하여 처리속도가 느리다.

③ 공개키 암호화의 대표적인 알고리즘에는 데이터 암호화 표준(Data Encryption Standard)이 있다.

④ 비밀키 암호화는 암호화와 복호화 과정에서 서로 다른 키를 사용하는 비대칭 암호화(asymmetric encryption)다.

정답 2번

간단한 해설

공개키 암호화는 비밀키를 공유하는 대칭키 암호화보다 **복잡한 수학적 연산**을 수행하므로, 일반적으로 같은 양의 데이터를 암호화할 때 처리속도가 느리다.

선지별 해설

① X: 공개키 암호화는 **서로 다른 공개키와 개인키**를 한 쌍으로 사용한다. 메시지의 기밀성을 보호하려면 송신자는 수신자의 공개키로 암호화하고, 수신자는 자신의 개인키로 복호화한다. 암호화와 복호화에 동일한 키를 사용하는 것은 대칭키 암호화이다.

② O: 여기서 비공개키 암호화는 비밀키를 공유하는 대칭키 암호화를 뜻한다. 공개키 암호화는 대칭키 암호화보다 수학적 연산의 부담이 커서, 일반적으로 **같은 양의 데이터를 처리하는 데 더 많은 시간**이 걸린다. 따라서 대용량 데이터의 암호화에는 처리속도가 빠른 대칭키 방식이 주로 쓰인다.

③ X: 데이터 암호화 표준인 DES(Data Encryption Standard)는 **대칭키 블록 암호 알고리즘**이다. 암호화와 복호화에 같은 비밀키를 사용한다. 공개키 암호화의 대표적인 알고리즘으로는 RSA가 있다.

④ X: 비밀키 암호화는 암호화와 복호화에 **동일한 키를 사용하는 대칭 암호화**이다. 서로 다른 공개키와 개인키를 한 쌍으로 사용하는 방식이 비대칭 암호화이며, 공개키 암호화라고도 한다.

NoteBase

문 12. CPU를 다른 프로세스로 교환하려면 이전 프로세스의 상태를 보관하고 새로운 프로세스의 보관된 상태로 복구하는 작업이 필요하다. 이 작업으로 옳은 것은?

① 세마포어(Semaphore)

② 모니터(Monitor)

③ 상호배제(Mutual Exclusion)

④ 문맥교환(Context Switching)

정답 4번

간단한 해설

이전 프로세스의 실행 상태를 저장하고 다음 프로세스의 저장된 상태를 복원하는 작업을 **문맥교환**이라고 한다.

공통 해설

문맥교환(Context Switching)은 CPU가 실행할 프로세스를 바꿀 때, 현재 프로세스의 실행 상태를 저장하고 다음 프로세스의 실행 상태를 복원하는 작업이다. 여기서 문맥은 프로그램 카운터와 CPU 레지스터 값 등 실행을 이어 가는 데 필요한 정보를 뜻한다.

현재 프로세스의 문맥을 저장해야 나중에 중단한 지점부터 실행을 재개할 수 있다. 다음 프로세스의 문맥을 복원하면 CPU는 그 프로세스가 저장해 둔 실행 위치와 레지스터 상태를 바탕으로 실행을 이어 간다. 문제에서 설명한 **이전 상태 보관과 다음 상태 복구**가 바로 문맥교환의 핵심 동작이다.

NoteBase

문 13. 응용프로그램 제작에 필요한 개발환경, SDK 등 플랫폼 자체를 서비스 형태로 제공하는 클라우드 컴퓨팅 서비스 모델은?

① DNS

② PaaS

③ SaaS

④ IaaS

정답 2번

간단한 해설

응용프로그램을 만들 수 있도록 **개발환경과 SDK 등 플랫폼을 제공**하는 모델은 PaaS이다. 사용자는 제공된 플랫폼 위에서 자신의 응용프로그램을 개발한다.

공통 해설

PaaS(Platform as a Service)는 응용프로그램의 개발과 실행에 필요한 플랫폼을 클라우드 서비스로 제공하는 모델이다. SDK는 소프트웨어 개발에 필요한 도구와 라이브러리 등을 묶은 개발 도구 모음으로, 개발자가 해당 환경에 맞는 응용프로그램을 만드는 데 사용한다.

문제에서 제공하는 대상은 개발환경과 SDK 등 **응용프로그램을 제작할 수 있는 기반**이다. 완성된 응용프로그램을 이용하거나 서버·저장장치 같은 인프라 자원만 제공받는 경우와 구별되며, 이러한 플랫폼 제공에 해당하는 명칭이 PaaS이다.

NoteBase

문 14. 다음 프로그램의 실행 결과로 옳은 것은?

```
#include <stdio.h>

int main(void)

{

    int array[] = {100, 200, 300, 400, 500};

    int *ptr;

    ptr = array;

    printf("%d\n", *(ptr+3) + 100);

}
```

① 200
② 300
③ 400
④ 500

정답 4번

간단한 해설

ptr + 3은 배열의 네 번째 원소를 가리키므로 ***(ptr+3)의 값은 400**이다. 여기에 100을 더한 500이 출력된다.

NoteBase

계산 과정

포인터의 초기 위치: array에는 100, 200, 300, 400, 500이 순서대로 저장되어 있다. 배열의 첨자는 0부터 시작하며, ptr = array를 실행하면 ptr은 첫 번째 원소인 array[0]의 주소를 저장한다.

포인터 이동: int형 포인터에 3을 더하면 **int형 원소 3개만큼 이동**한다. 따라서 ptr + 3은 array[3], 즉 네 번째 원소의 주소이다. 주소에 단순히 3바이트를 더하는 연산이 아니다.

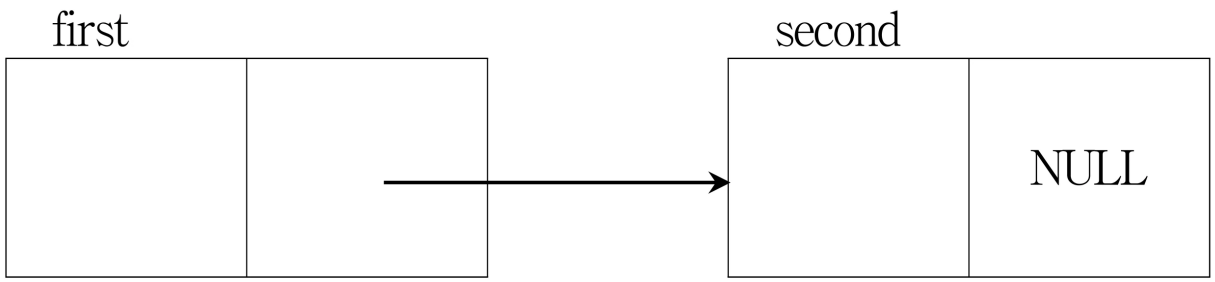
값 참조와 덧셈: *(ptr+3)에서 *는 해당 주소에 저장된 값을 읽는 역참조 연산자이다. array[3]의 값이 400이므로 *(ptr+3) + 100은 400 + 100 = 500이다. **100은 포인터가 아니라 읽어 온 정숫값에 더해진다.**

출력 결과: printf의 %d는 계산된 정숫값 500을 출력하고, 이어서 줄을 바꾼다. 이 결과는 ④의 500과 일치한다.

NoteBase

문 15. 다음 프로그램은 연결 리스트를 만들기 위한 코드의 일부분이다. 아래 그림과 같이 두 개의 노드 first, second가 연결되었다고 가정하고, 위의 코드를 참조하여 노드 tmp를 노드 first와 노드 second 사이에 삽입하고자 할 때, 프로그램 코드로 옳은 것은?

```
struct node {
  int number;
  struct node *link;
};
struct node first;
struct node second;
struct node tmp;
```



① tmp.link = &first;
first.link = &tmp;

② tmp.link = first.link;
first.link = &tmp;

③ tmp.link = &second;
first.link = second.link;

④ tmp.link = NULL;
second.link = &tmp;

NoteBase

정답 2번

간단한 해설

먼저 **tmp.link에 기존 first.link를 복사**하여 tmp가 second를 가리키게 한 뒤, first.link에 &tmp를 넣는다. 기존의 다음 노드 연결을 보존하면서 first → tmp → second로 연결을 바꾸는 방식이다.

핵심 풀이

구조체의 link는 다음 노드의 주소를 저장하는 포인터이다. 그림에서 first의 화살표가 second를 가리키므로 first.link에는 &second가 들어 있고, 마지막 노드인 second의 link는 NULL이다.

tmp.link = first.link;는 first.link에 저장된 **second의 주소를 tmp.link에 복사**한다. 이 시점에는 first와 tmp가 모두 second를 가리킨다. first.link 자체의 주소를 넣는 것이 아니라 그 안의 포인터 값을 복사하는 것이다.

first.link = &tmp;는 first가 다음 노드로 **tmp를 가리키도록** 바꾼다. &tmp는 tmp의 주소이므로, 두 문장을 실행하면 first → tmp → second → NULL이 된다. 이 두 대입문이 ②의 코드이다.

두 문장은 **기존 연결을 복사한 뒤 first.link를 변경**하는 순서로 실행한다. 순서를 뒤집으면 tmp.link = first.link;에서 이미 바뀐 &tmp를 복사하므로 tmp가 자기 자신을 가리키게 된다.

①처럼 tmp.link에 &first를 넣으면 first와 tmp가 서로를 가리키는 순환이 생긴다. ③처럼 first.link에 second.link를 넣으면 NULL이 저장되어 first에서 다음 노드로 이어지는 연결이 끊어진다. 사이에 삽입하려면 first는 tmp를, tmp는 second를 가리켜야 한다.

④는 tmp.link를 NULL로 두고 second.link에 &tmp를 넣으므로 first → second → tmp → NULL이 된다. 이는 두 노드 사이가 아니라 **second 뒤에 추가**하는 코드이다.

NoteBase

문 16. 다음 C 프로그램의 결과로 옳은 것은?

```
#include <stdio.h>
int main()
{
    int a, b;
    a = b = 1;

    if (a = 2)
       b = a + 1;
    else if (a == 1)
       b = b + 1;
    else
       b = 10;

    printf("%d, %d\n", a, b);
}
```

① 2, 3

② 2, 2

③ 1, 2

④ 2, 10

정답 1번

간단한 해설

조건식의 'a = 2'는 비교가 아니라 **대입**이며, 대입식의 값 2는 0이 아니므로 참으로 처리된다. 따라서 a는 2가 되고 첫 분기의 'b = a + 1'이 실행되어 출력은 2, 3이 된다.

NoteBase

계산 과정

초기값: a와 b는 정수형 변수이다. 'a = b = 1'은 오른쪽의 'b = 1'부터 적용되어 두 변수 모두 1이 된다.

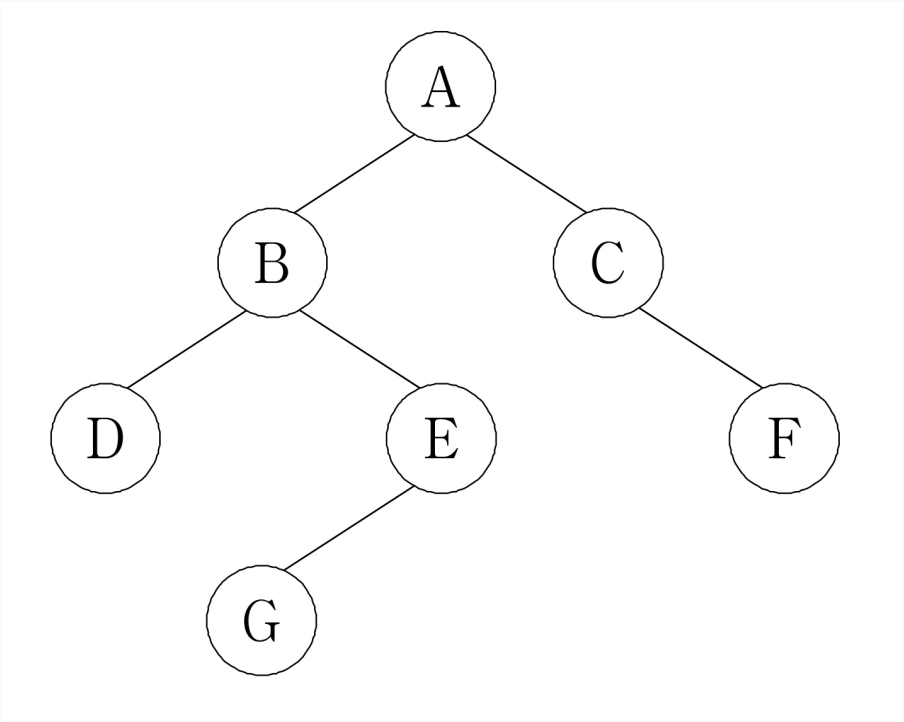
조건식 평가: 'a = 2'의 '='는 **값을 저장하는 대입 연산자**이고, '=='가 두 값이 같은지 비교하는 연산자이다. 따라서 이 조건식은 a에 2를 저장하며, 식의 값도 2가 된다. C의 if문은 **0이면 거짓, 0이 아니면 참**으로 판단하므로 첫 조건이 참이다.

분기 실행: 첫 조건이 참이므로 'b = a + 1'을 실행한다. 이때 a는 이미 2이므로 2 + 1 = 3을 b에 저장한다. if–else if–else는 앞에서 참인 분기가 실행되면 **나머지 분기를 건너뛴다**. 따라서 'a == 1'은 평가하지 않으며, 'b = b + 1'과 'b = 10'도 실행하지 않는다.

출력: printf는 두 %d 자리에 a와 b의 정수값을 차례로 넣어 출력한다. 최종적으로 a는 2, b는 3이므로 출력은 '2, 3'이며, ①의 값과 일치한다.

NoteBase

문 17. 다음 이진 트리에 대하여 후위 순회를 하는 경우 다섯 번째 방문하는 노드는?



① A

② C

③ D

④ F

정답 4번

간단한 해설

후위 순회는 **왼쪽 서브트리 → 오른쪽 서브트리 → 현재 노드** 순서로 방문한다. A의 왼쪽 서브트리에서 D·G·E·B를 방문한 뒤 오른쪽 서브트리의 F를 방문하므로, 다섯 번째 노드는 F이다.

NoteBase

계산 과정

후위 순회는 각 노드에서 **왼쪽 서브트리 → 오른쪽 서브트리 → 현재 노드** 순서를 재귀적으로 적용한다. 즉, 자식 쪽의 순회를 모두 마친 뒤 해당 노드를 방문하며, 자식이 없는 노드는 바로 방문한다.

A의 왼쪽 서브트리: B의 왼쪽 자식 D를 먼저 방문한다. 이어 B의 오른쪽 서브트리에서는 E보다 그 왼쪽 자식 G를 먼저 방문하므로 G → E가 된다. 두 서브트리를 마친 뒤 B를 방문하여 D → G → E → B 순서가 된다. 여기까지 방문한 노드는 4개이다.

A의 오른쪽 서브트리: C에는 왼쪽 자식이 없고 오른쪽 자식 F가 있으므로, **C보다 F를 먼저** 방문한다. 따라서 F가 다섯 번째, C가 여섯 번째이며, 양쪽 서브트리를 모두 마친 뒤 A를 일곱 번째로 방문한다.

전체 방문 순서는 D → G → E → B → F → C → A이다. 다섯 번째 방문 노드는 F로, ④에 해당한다.

NoteBase

문 18. 프로세스 스케줄링에 대한 설명으로 옳지 않은 것은?

① FCFS(First Come First Served) 스케줄링은 비선점 방식으로 대화식 시스템에 적합하다.

② SJF(Shortest Job First) 스케줄링은 실행 시간이 가장 짧은 작업(프로세스)을 신속하게 실행하므로 평균 대기시간이 FCFS 스케줄링보다 짧다.

③ Round-Robin 스케줄링은 우선순위가 적용되지 않은 단순한 선점형 방식이다.

④ 다단계 큐(Multilevel Queue) 스케줄링은 우선순위에 따라 준비 큐를 여러 개 사용하는 방식이다.

정답 1번

간단한 해설

FCFS는 먼저 도착한 작업이 CPU를 사용하는 비선점 방식이므로, 긴 작업 뒤에 도착한 짧은 요청도 오래 기다릴 수 있다. 따라서 **빠른 응답이 필요한 대화식 시스템에는 부적합하다**.

선지별 해설

① X: FCFS는 준비 큐에 도착한 순서대로 CPU를 배정하는 비선점 방식이다. 실행 중인 프로세스가 종료하거나 입출력 대기 등으로 CPU를 내놓을 때까지 다른 준비 프로세스가 CPU를 빼앗지 못한다. 긴 작업이 앞에 있으면 짧은 요청도 기다려야 하므로, '대화식 시스템에 적합하다'는 **'대화식 시스템에 부적합하다'**로 고쳐야 한다.

② O: SJF는 준비된 작업 중 CPU 실행 시간이 가장 짧은 작업부터 실행한다. 짧은 작업들이 긴 작업 뒤에서 기다리는 시간을 줄이므로 평균 대기시간을 줄이는 데 유리하다. 같은 시점에 준비된 작업들의 실행 시간을 알고 비교하면 **짧은 작업부터 실행하는 순서가 평균 대기시간을 최소화**한다. 다만 FCFS의 실행 순서가 이미 SJF와 같다면 두 방식의 평균 대기시간도 같다.

③ O: 기본 Round-Robin은 프로세스별 우선순위를 구별하지 않고 준비 큐를 순환하며 CPU를 배정한다. 각 프로세스는 정해진 시간 할당량만큼 실행할 수 있고, **할당량을 소진하면 선점**되어 준비 큐의 뒤로 이동한다. 이처럼 CPU를 번갈아 사용하게 하는 단순한 선점형 방식이다.

④ O: 다단계 큐는 우선순위나 작업 특성에 따라 프로세스를 구분하여 **여러 개의 준비 큐**로 관리한다. 예를 들어 대화식 작업과 일괄처리 작업을 서로 다른 큐에 둘 수 있다. 큐 사이에 우선순위를 적용할 수 있으며, 각 큐 내부에는 RR이나 FCFS 등 서로 다른 스케줄링 방식을 적용할 수 있다.

NoteBase

문 19. TCP/IP 프로토콜 스택에 대한 설명으로 옳은 것은?

① 데이터링크(datalink) 계층, 전송(transport) 계층, 세션(session) 계층 및 응용(application) 계층으로 구성된다.

② ICMP는 데이터링크 계층에서 사용 가능한 프로토콜이다.

③ UDP는 전송 계층에서 사용되는 비연결형 프로토콜이다.

④ 응용 계층은 데이터가 목적지까지 찾아갈 경로를 설정하기 위해 라우팅(routing) 프로토콜을 운영한다.

정답 3번

간단한 해설

UDP는 전송 계층에서 동작하며, 데이터를 보내기 전에 상대와 연결을 설정하지 않는 **비연결형 프로토콜**이다.

선지별 해설

① X: TCP/IP의 4계층은 네트워크 접근 계층, 인터넷 계층, 전송 계층, 응용 계층이다. 선지에는 **인터넷 계층이 빠지고 세션 계층이 들어가 있다**. 네트워크 접근 계층은 직접 연결된 네트워크에서의 통신을 담당하며, OSI 모형의 세션 계층 기능은 TCP/IP에서는 응용 계층에 포함된다.

② X: ICMP는 데이터링크 계층이 아니라 **인터넷 계층**에 속하는 제어 프로토콜이다. IP 패킷으로 메시지를 전달하며, 목적지에 도달할 수 없다는 오류 등을 알리는 데 사용된다.

③ O: UDP는 전송 계층에서 응용 프로그램 사이의 데이터 전달을 지원한다. TCP와 달리 **연결 설정 절차 없이** 데이터그램을 전송하는 비연결형 프로토콜이다. UDP 자체는 데이터의 도착이나 도착 순서를 보장하지 않는다.

④ X: 목적지까지의 경로 선택과 IP 패킷 전달은 **인터넷 계층의 기능**이다. 응용 계층은 파일 전송이나 전자우편처럼 응용 프로그램에 필요한 통신 서비스를 제공하며, 대표적인 프로토콜로 FTP와 SMTP가 있다.

NoteBase

문 20. 다음 테이블 인스턴스(Instance)들에 대하여 오류 없이 동작하는 SQL(Structured Query Language) 문장은?

STUDENT

칼럼 이름	데이터 타입	키 타입	설명
studno	숫자	기본키	학번
name	문자열		이름
grade	숫자		학년
height	숫자		키
deptno	숫자		학과 번호

PROFESSOR

칼럼 이름	데이터 타입	키 타입	설명
profno	숫자	기본키	번호
name	문자열		이름
position	문자열		직급
salary	숫자		급여
deptno	숫자		학과 번호

①
```
SELECT deptno, position, AVG(salary)
FROM PROFESSOR
GROUP BY deptno;
```

②
```
(SELECT studno, name
FROM STUDENT
WHERE deptno = 101)
UNION
(SELECT profno, name
FROM PROFESSOR
WHERE deptno = 101);
```

③
```
SELECT grade, COUNT(*), AVG(height)
FROM STUDENT
WHERE COUNT(*) > 2
GROUP BY grade;
```

④
```
SELECT name, grade, height
FROM STUDENT
WHERE height > (SELECT height, grade
FROM STUDENT
WHERE name = '홍길동');
```

NoteBase

정답 2번

간단한 해설

UNION은 두 조회 결과의 **열 개수가 같고 대응하는 열의 자료형이 호환**되면 사용할 수 있다. studno와 profno는 모두 숫자형이고 두 name은 모두 문자열형이므로, 학과 번호가 101인 학생과 교수의 번호·이름을 하나의 결과로 합칠 수 있다.

선지별 해설

① X: GROUP BY deptno는 학과별로 행을 묶고, AVG(salary)는 각 학과의 평균 급여를 구한다. 그런데 SELECT의 position은 집계 함수로 처리하지 않았고 그룹 기준에도 없다. 한 학과에 여러 직급이 있으면 그중 어느 직급을 출력할지 정할 수 없다. 학과·직급별 평균 급여를 구하려면 **GROUP BY deptno, position**으로 고친다.

② O: UNION은 두 SELECT 결과를 위아래로 합치는 집합 연산이다. 두 조회 모두 열이 2개이며, 첫 번째 열인 studno와 profno는 숫자형, 두 번째 열인 name은 문자열형이므로 **대응하는 열의 자료형이 호환**된다. 첫 번째 열의 이름이 서로 달라도 사용할 수 있다. 각 WHERE deptno = 101은 해당 테이블에서 학과 번호가 101인 행을 선택하며, UNION은 합친 결과에서 두 열의 값이 모두 같은 중복 행을 제거한다.

③ X: WHERE는 그룹을 만들기 전 개별 행을 걸러 내는 절이므로, 그룹의 행 수를 구하는 COUNT(*)를 이 위치의 조건에 사용할 수 없다. 학년별 인원이 2명을 초과하는 그룹을 고르려면 **그룹 조건을 HAVING에** 써야 한다. FROM STUDENT 뒤를 GROUP BY grade HAVING COUNT(*) > 2로 고치면, 해당 학년의 인원수와 평균 키를 조회할 수 있다.

④ X: height > (...)는 하나의 키 값과 부질의 결과를 비교하는 형태인데, 부질의가 height와 grade라는 **두 열을 반환**하므로 비교 대상이 맞지 않는다. 또한 name은 기본키가 아니어서 이름이 '홍길동'인 행이 하나라고 보장할 수 없다. 단일 값과 비교하려면 부질의가 한 열의 값 하나를 반환하도록 해야 한다. 예를 들어 이름이 '홍길동'인 학생들의 평균 키보다 큰 학생을 조회하려면, 부질의를 SELECT AVG(height) FROM STUDENT WHERE name = '홍길동'으로 바꾼다. GROUP BY 없는 AVG는 하나의 집계값을 반환하며, 비교할 값이 NULL이면 해당 조건은 참이 되지 않는다.

NoteBase