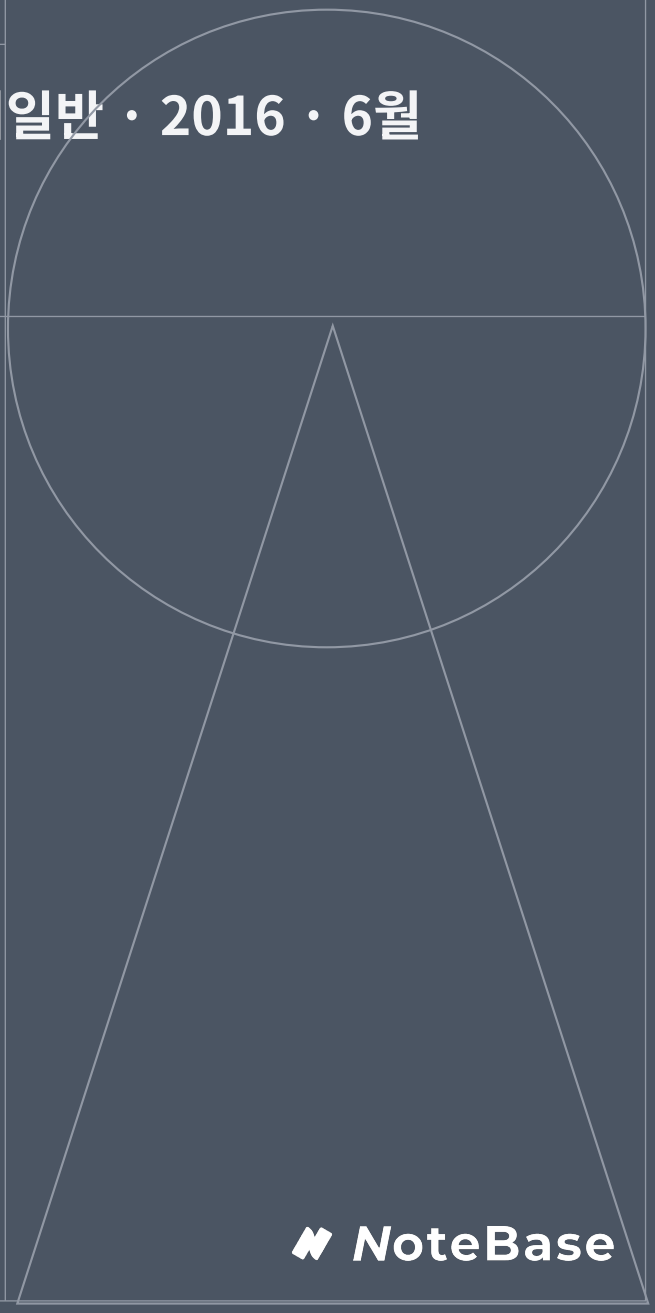
기출OX 해설집

공무원/서울시/9급 · 컴퓨터일반 · 2016 · 6월



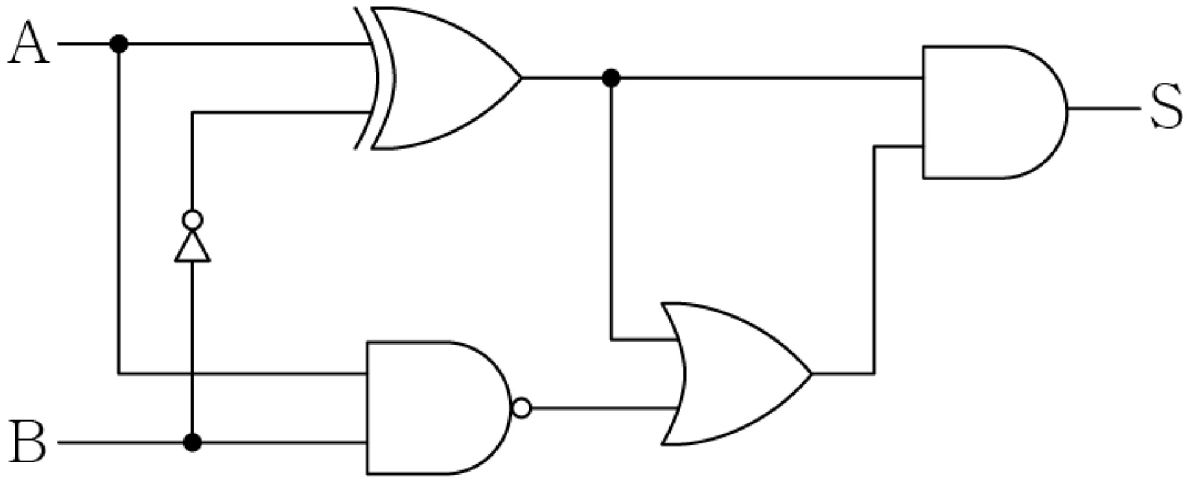
NoteBase

기출OX 해설집

공무원/서울시/9급 · 컴퓨터일반 · 2016 · 6월

공무원/서울시/9급 · 컴퓨터일반 · 2016 · 6월

문 1. 다음 논리회로에서 $A = 1010, B = 0010$일 때, S에 출력되는 값은?



① 1011

② 1101

③ 0111

④ 1110

정답 3번

간단한 해설

B를 반전한 1101과 A=1010을 **XOR 연산**하면 0111이 된다. 마지막 AND 게이트의 다른 입력은 1111이므로 출력은 **0111**로 유지된다.

NoteBase

계산 과정

A=1010, B=0010의 같은 자리 비트끼리 논리 연산을 적용한다. 각 자리는 독립적으로 처리하며, 앞자리의 0도 포함하여 네 자리로 표시한다.

B의 NOT 연산: B에서 위로 갈라진 경로의 삼각형과 출력 쪽 작은 원은 NOT 게이트이다. **0과 1을 서로 바꾸므로** 0010은 1101이 된다.

위쪽 XOR 연산: 입력 쪽에 곡선이 하나 더 있는 게이트는 XOR 게이트이다. A=1010과 반전된 B=1101을 입력받아 **두 비트가 다를 때만 1**을 출력한다. 왼쪽부터 (1, 1), (0, 1), (1, 0), (0, 1)이므로 출력은 0111이다.

아래쪽 NAND 연산: A와 B가 직접 들어가는 아래 게이트는 AND 출력에 작은 원이 붙은 NAND 게이트이다. 두 비트가 모두 1일 때만 1인 AND 결과를 반전한다. 1010 AND 0010 = 0010이므로, **이를 반전한 NAND 출력**은 1101이다.

가운데 OR 연산: 위쪽 XOR 출력 0111과 아래쪽 NAND 출력 1101이 OR 게이트로 들어간다. **둘 중 하나라도 1이면 1**이므로 0111 OR 1101 = 1111이다.

마지막 AND 연산: 위쪽 XOR 출력 0111은 마지막 AND 게이트에도 직접 연결된다. 다른 입력은 OR 출력 1111이다. AND는 두 비트가 모두 1일 때만 1이므로, **1111과 AND 연산하면 상대 입력이 그대로 유지**되어 S=0111이 된다.

출력값 0111은 ③의 값과 일치한다.

NoteBase

문 2. 현재 실행 중인 프로세스에 할당된 CPU사용권을 다른 프로세스에게 할당하려면, 현재 실행 중인 프로세스의 실행 정보를 저장하고 다음으로 실행할 프로세스의 실행정보를 가져오는 과정이 필요하다. 이 과정을 무엇이라고 하는가?

① 컨텍스트 스위칭(Context Switching)

② 가상메모리(Virtual Memory)

③ 교체정책(Replacement Strategy)

④ 디스패치(Dispatch)

정답 1번

간단한 해설

프로세스를 바꿀 때 현재 프로세스의 실행 정보를 저장하고 다음 프로세스의 실행 정보를 복원하는 과정을 **컨텍스트 스위칭**이라고 한다. 저장한 정보가 있어야 중단된 프로세스도 나중에 이어서 실행할 수 있다.

공통 해설

컨텍스트는 프로세스의 실행을 이어 가는 데 필요한 정보로, 다음에 실행할 명령어의 위치를 나타내는 프로그램 카운터와 CPU 레지스터 값 등이 포함된다. 컨텍스트 스위칭은 **현재 프로세스의 컨텍스트를 저장하고 다음 프로세스의 컨텍스트를 복원**하여 CPU가 다른 프로세스를 실행하도록 바꾸는 과정이다.

예를 들어 프로세스 A를 실행하다가 B로 전환한다면, A의 프로그램 카운터와 레지스터 값 등을 저장한 뒤 B가 이전에 저장해 둔 값을 CPU에 복원한다. 나중에 A로 돌아올 때는 A의 값을 복원하므로 중단한 지점부터 실행을 계속할 수 있다. 문제의 '실행 정보를 저장하고 다음 프로세스의 실행 정보를 가져오는 과정'이 바로 이 저장·복원에 해당한다.

디스패치는 스케줄러가 선택한 프로세스에 CPU 제어권을 넘기는 것을 뜻한다. 그 과정에서 컨텍스트 스위칭이 수행될 수 있지만, 문제에서 구체적으로 설명한 **실행 정보의 저장·복원**을 가리키는 명칭은 컨텍스트 스위칭이다.

NoteBase

문 3. 다음 중 유효한 SQL 문장이 아닌 것은?

① SELECT * FROM Lawyers WHERE firmName LIKE '% and %';

② SELECT firmLoc, COUNT(*) FROM Firms WHERE employees < 100;

③ SELECT COUNT(*) FROM Firms WHERE employees < 100;

④ SELECT firmLoc, SUM(employees) FROM Firms GROUP BY firmLoc WHERE SUM(employees) < 100;

정답 4번

간단한 해설

④의 SUM(employees) < 100은 개별 행이 아니라 **그룹별 집계 결과에 대한 조건**이므로 HAVING에 써야 한다. WHERE는 GROUP BY보다 앞에 오며, 집계하기 전의 행을 거른다.

선지별 해설

① O: LIKE는 문자열이 주어진 패턴과 일치하는지 검사한다. %는 길이가 0 이상인 임의의 문자열을 뜻하므로, '% and %'는 **앞뒤 공백을 포함한 ' and '**가 들어 있는 firmName을 찾는다. SELECT *는 조건을 만족하는 Lawyers 행의 모든 열을 조회한다.

② O: **SQLite처럼 비집계 열을 함께 조회하는 확장을 허용하는 DBMS에서는 유효하다.** WHERE로 employees가 100 미만인 행을 고른 뒤, COUNT(*)는 그 행 전체의 개수를 센다. GROUP BY가 없으므로 지역별로 집계하지 않으며, SQLite에서 firmLoc은 임의의 한 행에서 가져온 값이다. 다만 비집계 열의 그룹화 규칙을 엄격히 적용하는 DBMS에서는 허용되지 않으므로 모든 SQL 환경에 통용되는 문장은 아니다. 지역별 개수를 구하려면 문장 끝의 세미콜론 앞에 GROUP BY firmLoc을 추가한다.

③ O: WHERE employees < 100으로 조건에 맞는 행을 먼저 고른 뒤, **COUNT(*)로 그 행의 개수**를 구한다. 일반 열을 함께 조회하지 않고 집계값 하나만 반환하므로 GROUP BY 없이 사용할 수 있다. 조건을 만족하는 행이 없으면 개수는 0이다.

④ X: WHERE를 GROUP BY 뒤에 둔 것은 절의 작성 순서에 맞지 않는다. 또한 SUM(employees)는 지역별로 묶은 뒤 계산하는 합계이므로 그 합계의 조건은 **WHERE가 아니라 HAVING**에 둔다. 다음과 같이 고친다: SELECT firmLoc, SUM(employees) FROM Firms GROUP BY firmLoc HAVING SUM(employees) < 100; 이 문장은 같은 firmLoc끼리 직원 수를 합산하고, 그 합계가 100 미만인 지역만 조회한다.

NoteBase

문 4. 다음 중 나머지 셋과 역할 기능이 다른 하나는?

① Array processor

② DMA

③ GPU

④ SIMD

정답 2번

간단한 해설

DMA는 CPU가 데이터를 일일이 옮기지 않고 **입출력장치와 메모리 사이에서 직접 전송**하도록 하는 방식이다. 나머지 세 항목은 여러 데이터를 병렬로 연산하는 것과 관련된다.

선지별 해설

① Array processor: 배열 처리기는 여러 처리 요소를 이용하여 배열이나 벡터의 여러 데이터에 대한 **연산을 병렬로 수행**한다. 많은 데이터에 같은 연산을 적용하는 작업을 빠르게 처리하는 데 쓰인다.

② DMA: Direct Memory Access, 즉 직접 메모리 접근은 **입출력장치와 메모리 사이의 데이터 전송**을 위한 방식이다. DMA 제어기가 전송을 담당하여 CPU가 데이터를 하나씩 직접 옮기는 부담을 줄인다. 여러 데이터의 산술·논리 연산을 병렬로 수행하는 방식과는 역할이 다르다.

③ GPU: 그래픽 처리 장치는 많은 연산 자원을 이용하여 여러 데이터에 대한 **계산을 병렬로 처리**한다. 그래픽 처리뿐 아니라 병렬화할 수 있는 대규모 계산에도 활용되므로, 연산의 병렬 처리와 관련된다.

④ SIMD: Single Instruction Multiple Data는 **하나의 명령을 여러 데이터에 동시에 적용**하는 병렬 처리 방식이다. 처리하는 데이터는 서로 다르지만 같은 연산을 수행한다는 점이 핵심이다.

①·③·④는 데이터에 대한 병렬 연산과 관련되고, ② DMA는 입출력 데이터 전송을 담당한다는 점에서 나머지 셋과 역할이 다르다.

NoteBase

문 5. 다음은 IPv6에 대한 설명이다. 옳지 않은 것은?

① 기존의 IP 주소 공간이 빠른 속도로 고갈되어 왔기 때문에 고안되었다.

② IPv6는 IP 주소 크기를 기존의 4바이트에서 6바이트로 확장했다.

③ IPv6는 유니캐스트, 멀티캐스트 주소뿐만 아니라 새로운 주소 형태인 애니캐스트 주소가 도입되었다.

④ 네트워크 프로토콜을 바꾼다는 것은 매우 어렵기 때문에 IPv6로의 전환을 위해 여러 방법들이 고안되었다.

정답 2번

간단한 해설

②의 '6바이트'는 **16바이트(128비트)**로 고쳐야 한다. IPv6는 IPv4의 4바이트(32비트) 주소를 확장하여 훨씬 넓은 주소 공간을 제공한다.

선지별 해설

① O: IPv4는 32비트 주소를 사용하므로 주소 공간에 한계가 있다. 인터넷에 연결되는 장치가 늘면서 주소 부족 문제가 커졌고, 이를 해결하기 위해 **주소 공간을 크게 확장한 IPv6**가 고안되었다.

② X: IP 주소의 길이는 IPv4에서 4바이트(32비트), IPv6에서 **16바이트(128비트)**이다. 1바이트는 8비트이므로 $16 \times 8 = 128$비트이다. 따라서 '4바이트에서 6바이트로 확장했다'는 표현은 '4바이트에서 16바이트로 확장했다'로 고쳐야 한다.

③ O: IPv6는 유니캐스트·멀티캐스트·애니캐스트 주소를 정의한다. 유니캐스트는 하나의 인터페이스에, 멀티캐스트는 해당 그룹의 모든 인터페이스에 패킷을 전달한다. 애니캐스트는 같은 주소를 공유하는 여러 인터페이스 중 **라우팅 기준으로 가장 가까운 하나**에 전달한다. 여기서 가깝다는 것은 물리적 거리가 아니라 라우팅 프로토콜이 판단하는 경로상의 거리를 뜻한다.

④ O: IPv4만 지원하는 장비는 IPv6 패킷을 직접 처리할 수 없고, 기존 네트워크 전체를 한꺼번에 교체하기도 어렵다. 이를 고려하여 **점진적인 전환을 위한 방법**들이 고안되었다. 예를 들어 듀얼 스택은 한 장비에서 IPv4와 IPv6를 함께 지원하는 방식이고, 터널링은 IPv6 패킷을 IPv4 패킷 안에 넣어 기존 IPv4 네트워크를 통과시키는 방식이다.

NoteBase

문 6. 다음 정렬 알고리즘 중 최악의 경우에 시간복잡도가 가장 낮은 것은?

① 버블 정렬(Bubble sort)

② 삽입 정렬(Insertion sort)

③ 퀵 정렬(Quick sort)

④ 힙 정렬(Heap sort)

정답 4번

간단한 해설

힙 정렬은 최악의 경우에도 시간복잡도가 $O(n\log n)$으로 보장된다. 다른 세 정렬은 최악의 경우 $O(n^2)$이므로, 자료 수가 커질 때 **힙 정렬의 최악 시간 증가율이 더 낮다**.

선지별 해설

자료 수를 n이라 할 때, 최악 시간복잡도는 같은 크기의 입력 중 처리 시간이 가장 많이 드는 경우의 증가율을 나타낸다. 평균적인 성능이 아니라 최악의 입력에서 필요한 연산량을 비교한다.

① 버블 정렬: 최악 시간복잡도는 $O(n^2)$이다. 인접한 두 원소를 비교하여 교환하는 과정을 반복한다. 역순으로 나열된 자료처럼 교환이 많이 필요한 입력에서는 **여러 차례에 걸쳐 배열을 훑어야** 하므로 비교 횟수가 자료 수의 제곱에 비례한다.

② 삽입 정렬: 최악 시간복잡도는 $O(n^2)$이다. 각 원소를 앞쪽의 정렬된 부분에 삽입한다. 역순 입력에서는 새 원소를 삽입할 때마다 **앞의 원소들을 모두 한 칸씩 이동**해야 하므로, 전체 비교·이동 횟수가 자료 수의 제곱에 비례한다.

③ 퀵 정렬: 최악 시간복잡도는 $O(n^2)$이다. 피벗을 기준으로 자료를 나누는데, 피벗이 매번 최솟값이나 최댓값으로 선택되면 **분할이 한쪽으로 치우친다**. 이때 처리할 자료 수가 한 개씩만 줄어드는 분할을 반복하므로 전체 연산량이 자료 수의 제곱에 비례한다. 평균 시간복잡도 $O(n\log n)$과 구별해야 한다.

④ 힙 정렬: 최악 시간복잡도는 $O(n\log n)$이다. 최대 힙을 구성한 뒤 루트의 최댓값을 정렬 영역으로 옮기고, 남은 부분의 힙 조건을 복구하는 과정을 반복한다. 힙은 완전이진트리이므로 높이가 $O(\log n)$이고, **한 번의 힙 복구는 트리 높이만큼의 작업**으로 끝난다. 초기 힙 구성은 $O(n)$, 반복되는 추출·복구는 전체 $O(n\log n)$이므로 최악에도 이 시간복잡도를 보장한다.

최악 시간복잡도를 비교하면 버블·삽입·퀵 정렬의 $O(n^2)$보다 힙 정렬의 $O(n\log n)$이 낮으므로, 문항의 조건에 해당하는 것은 ④의 힙 정렬이다.

NoteBase

문 7. 다음 C 프로그램의 실행 결과는?

```
#include<stdio.h>
int main()
{
    char* array1[2]={"Good morning", "C language"};
     printf("%s ₩n", array1[0]+5);
     printf("%c ₩n", *(array1[1]+6));
     return 0;
}
```

① Good morning
C-language

② morning
a

③ morning
g

④ morning
u

정답 4번

간단한 해설

array1[0]+5는 첫 문자열에서 **인덱스 5의 m부터 시작하는 문자열**을 가리킨다. 두 번째 문자열은 공백도 한 문자로 세므로 인덱스 6의 문자는 **u**이다.

NoteBase

계산 과정

array1은 문자 포인터 두 개를 저장하는 배열이다. array1[0]은 "Good morning"의 첫 문자 G를, array1[1]은 "C language"의 첫 문자 C를 가리킨다. 문자 포인터에 정수를 더하면 그 수만큼 뒤의 문자 위치로 이동하며, **첫 문자의 인덱스는 0**이다.

첫 번째 printf: array1[0]+5는 "Good morning"에서 G(0), o(1), o(2), d(3), 공백(4)을 지나 m(5)을 가리킨다. %s는 전달받은 주소에서 시작하여 문자열의 끝을 나타내는 널 문자 전까지 출력하므로, **morning 전체**가 출력된다.

두 번째 printf: "C language"의 문자는 C(0), 공백(1), l(2), a(3), n(4), g(5), u(6), a(7), g(8), e(9) 순서이다. array1[1]+6은 u의 주소이고, 앞의 *는 **그 주소에 저장된 문자 하나**를 읽는다. 따라서 %c로 u가 출력된다.

두 printf가 각각 줄을 바꾸므로 첫째 줄의 내용은 morning, 둘째 줄의 내용은 u이다. 이는 ④의 출력 결과와 일치한다.

NoteBase

문 8. 메모리 크기가 200KB인 시스템에서 요구 페이징(demand paging)으로 가상 메모리(virtual memory)를 구현한다고 하자. 페이지 크기가 2KB이고 페이지 테이블(page table)의 각 항목이 3바이트라고 하면, 25KB 크기의 프로세스를 위한 최소 페이지 테이블의 크기는 어떻게 되는가?

① 25바이트

② 39바이트

③ 60바이트

④ 75바이트

정답 2번

간단한 해설

25KB를 2KB씩 나누면 마지막 1KB에도 페이지 하나가 필요하므로 **총 13개의 페이지**가 필요하다. 페이지마다 3바이트짜리 항목을 두므로 최소 페이지 테이블 크기는 39바이트이다.

계산 과정

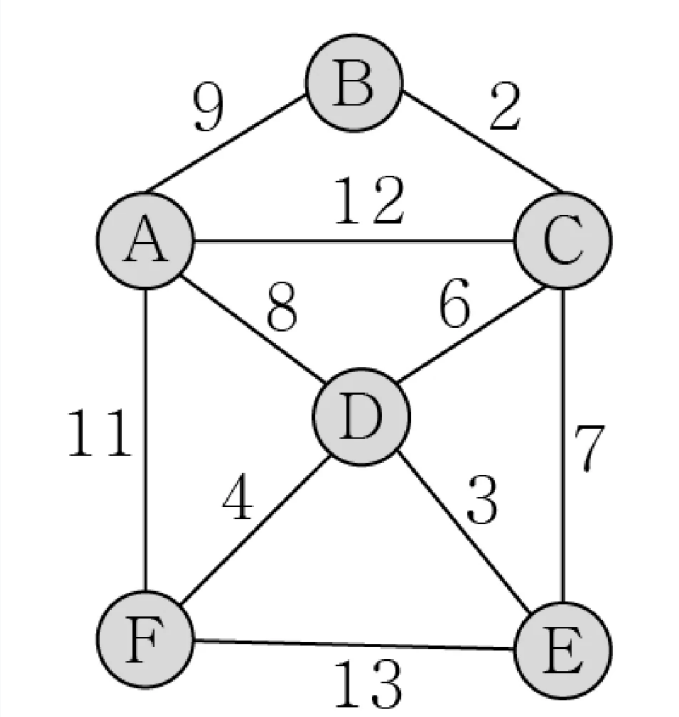
페이지 테이블은 프로세스의 가상 페이지와 물리 메모리 프레임의 대응 및 상태를 관리하며, 이 문제에서는 **프로세스의 페이지마다 항목 하나**가 필요하다. 요구 페이징으로 현재 메모리에 올라온 페이지가 일부여도 프로세스의 전체 페이지에 대한 항목이 필요하므로, 시스템의 물리 메모리 크기인 200KB를 기준으로 항목 수를 구하지 않는다.

필요한 페이지 수: 프로세스 크기 25KB를 페이지 크기 2KB로 나눈 뒤 올림한다. 따라서 $\left\lceil \frac{25}{2} \right\rceil = 13$개이다. 12개 페이지에는 24KB만 담기므로, 남은 1KB를 담을 **마지막 페이지 하나가 더 필요**하다.

페이지 테이블 크기: 페이지가 13개이므로 항목도 13개이다. 항목 하나가 3바이트이므로 $13 \times 3 =$ 39바이트이며, 이는 ②의 39바이트에 해당한다.

NoteBase

문 9. 다음 그래프에서 최소 비용의 '신장 트리' 값은 얼마인가?



① 16

② 20

③ 23

④ 26

정답 3번

간단한 해설

최소 비용 신장 트리는 모든 정점을 연결하되 **사이클을 만들지 않는** 간선들의 비용 합을 최소로 한다. 이 그래프에서는 비용 2, 3, 4, 6, 8인 간선을 선택하여 총비용이 23이 된다.

NoteBase

계산 과정

신장 트리는 그래프의 모든 정점을 연결하면서 사이클이 없는 부분 그래프이다. 정점이 A~F의 6개이므로 간선 5개로 연결한다. 최소 비용을 구하려면 크루스칼 알고리즘에 따라 **비용이 작은 간선부터, 사이클을 만들지 않을 때만** 선택한다.

먼저 B–C(2), D–E(3), D–F(4)를 선택한다. 아직 사이클은 없으며, B·C가 한 묶음, D·E·F가 다른 묶음으로 연결되어 있다.

다음으로 C–D(6)를 선택하면 두 묶음이 합쳐져 B·C·D·E·F가 연결된다. 선택한 간선은 4개이고 비용 합은 2 + 3 + 4 + 6 = 15이지만, A는 아직 연결되지 않았다.

그다음 C–E(7)는 선택하지 않는다. 이미 C–D–E 경로가 있으므로 이 간선을 추가하면 **C–D–E–C의 사이클**이 생긴다.

이어서 A–D(8)를 선택한다. A와 기존 묶음을 연결하는 간선 중 A–D(8)가 A–B(9), A–F(11), A–C(12)보다 저렴하다. 이제 모든 정점이 간선 5개로 연결되므로 선택을 끝낸다.

선택한 간선의 총비용은 2 + 3 + 4 + 6 + 8 = 23이다. 이는 ③의 23에 해당한다.

NoteBase

문 10. 라우팅 알고리즘은 라우터에 패킷이 도착했을 때 포워딩 테이블을 검색하고 패킷이 전달될 인터페이스를 결정하는 알고리즘이다. 다음 중 라우팅 알고리즘이 아닌 것은?

① RIP(Routing Information Protocol)

② OSPF(Open Shortest Path First)

③ CDMA(Code Division Multiple Access)

④ BGP(Border Gateway Protocol)

정답 3번

간단한 해설

CDMA는 사용자마다 서로 다른 코드를 부여하여 같은 통신 자원을 공유하게 하는 **코드 분할 다중 접속 방식**이다. 패킷이 목적지까지 갈 경로를 정하는 라우팅 방식에 해당하지 않는다.

공통 해설

라우팅은 패킷을 목적지까지 전달할 **경로를 결정**하는 기능이다. RIP, OSPF, BGP는 라우터 사이에서 경로 정보를 교환하고 전달 경로를 정하는 데 사용하는 라우팅 프로토콜이다. 이 문항에서는 이들을 라우팅 알고리즘에 해당하는 후보로 묶는다.

CDMA는 여러 사용자가 같은 주파수 대역에서 동시에 통신할 때 **사용자별 코드를 이용해 신호를 구별**하는 다중 접속 방식이다. 네트워크에서 패킷의 전달 경로를 선택하는 기능이 아니므로, 라우팅 알고리즘이 아닌 것을 묻는 조건에 해당한다.

NoteBase

문 11. 암달(Amdahl)의 법칙은 컴퓨터 시스템의 일부를 개선할 때 전체적으로 얼마만큼의 최대 성능 향상을 기대할 수 있는지를 예측하는 데 사용된다. 만약 특정 응용프로그램의 75%가 멀티코어(Multicore)를 이용한 병렬 수행이 가능하고 나머지 25%는 코어의 수가 증가해도 순차 실행만 가능하다는 전제 하에, 컴퓨팅 코어(Core)의 수를 4개로 늘릴 때 기대할 수 있는 최대 성능 향상은 약 몇 배인가?

① 약 1.28배

② 약 2.28배

③ 약 3.28배

④ 약 4.28배

정답 2번

간단한 해설

순차 실행 부분의 시간은 그대로이고, **병렬 실행 부분의 시간만 4분의 1**로 줄어든다. 전체 실행 시간이 기존의 0.4375배가 되므로 최대 성능 향상은 약 2.28배이다.

계산 과정

암달의 법칙은 개선되지 않는 부분의 실행 시간과 개선되는 부분의 실행 시간을 더해 전체 성능 향상을 구한다. 기존의 단일 코어 실행 시간을 1로 놓고, 병렬 처리의 추가 지연 없이 작업을 네 코어에 고르게 나누는 이상적인 경우를 기준으로 한다.

순차 실행 부분은 전체의 25%이므로 코어를 늘려도 실행 시간이 0.25로 유지된다. 병렬 실행 부분은 전체의 75%이며, 네 코어가 나누어 처리하므로 실행 시간은 $\frac{0.75}{4} = 0.1875$가 된다. **전체 시간이 아니라 병렬 실행 부분의 시간만** 코어 수로 나눈다.

4개 코어를 사용할 때의 전체 실행 시간은 0.25 + 0.1875 = 0.4375이다. 이는 같은 작업을 끝내는 데 기존 실행 시간의 43.75%가 걸린다는 뜻이다.

같은 작업의 성능 향상 배수는 **기존 실행 시간을 개선 후 실행 시간으로 나눈 값**이다. 따라서 $\frac{1}{0.4375} = \frac{16}{7} = 2.2857\ldots$배이며, 선지 ②의 약 2.28배에 해당한다.

NoteBase

문 12. <보기> 중 우리가 흔히 인터넷을 통해 비용을 지불하거나 혹은 무료로 사용하는, 클라우드 저장 서버에 대한 분류로 옳은 것을 모두 고르면?

<보기>

ㄱ. Public cloud

ㄴ. Private cloud

ㄷ. Software as a service(Saas)

ㄹ. Platform as a service(Paas)

ㅁ. Infrastructure as a service(Iaas)

① ㄱ, ㄷ

② ㄱ, ㅁ

③ ㄴ, ㄷ

④ ㄴ, ㄹ

정답 2번

간단한 해설

일반 대중에게 개방된 클라우드는 **Public cloud**이다. 이 문항처럼 저장 서버의 공간을 기반 자원으로 제공하는 측면에서 분류하면 **IaaS**에 해당한다.

NoteBase

보기별 해설

클라우드는 이용 대상에 따라 배치 모델을, 제공하는 자원의 수준에 따라 서비스 모델을 구분한다. 따라서 Public cloud와 IaaS는 서로 경쟁하는 분류가 아니라 한 서비스에 함께 적용할 수 있는 분류이다.

ㄱ. Public cloud: **일반 대중이 이용할 수 있도록 개방**된 클라우드이다. 문제의 일반 이용자가 인터넷으로 사용하는 공개 서비스에 해당하며, 유료인지 무료인지는 이 분류를 결정하지 않는다.

ㄴ. Private cloud: **특정 조직이 전용으로 이용**하는 클라우드이다. 일반 대중에게 개방된 문제의 서비스는 Public cloud로 분류한다. 개인 계정으로 접속하거나 자신의 파일을 비공개로 저장한다는 이유만으로 Private cloud가 되는 것은 아니다.

ㄷ. Software as a service(Saas): 제공자가 운영하는 **완성된 응용 소프트웨어**를 이용하는 서비스 모델이다. 이 문항은 파일 관리 앱의 기능보다 저장 서버의 공간을 기반 자원으로 제공한다는 측면에서 IaaS로 분류한다. 다만 완성된 파일 관리·공유 앱을 제공하는 서비스는 SaaS로 분류할 수 있으므로, 저장 기능이 있다는 이유만으로 모든 서비스를 IaaS라고 단정해서는 안 된다.

ㄹ. Platform as a service(Paas): 이용자가 응용 프로그램을 개발·배포할 수 있도록 **개발 및 실행 환경**을 제공하는 서비스 모델이다. 문제는 프로그램을 개발할 플랫폼이 아니라 저장 자원의 제공을 묻고 있으므로 IaaS에 해당한다.

ㅁ. Infrastructure as a service(Iaas): 서버, 저장 공간, 네트워크 등 **기본 컴퓨팅 자원**을 제공하는 서비스 모델이다. 저장 서버의 공간을 인터넷으로 제공받아 사용하는 측면이 이에 해당한다.

해당하는 분류는 ㄱ과 ㅁ이며, 이를 묶은 선택지는 ②이다.

NoteBase

문 13. 16진수로 표현된 $B9E_{(16)}$를 2진수로 표현하면 다음 중 무엇인가?

① $1100\ 0101\ 1101_{(2)}$

② $0101\ 0101\ 1001_{(2)}$

③ $1011\ 1001\ 1110_{(2)}$

④ $1110\ 0101\ 1101_{(2)}$

정답 3번

간단한 해설

16진수 한 자리는 **2진수 4자리**에 대응한다. B·9·E를 각각 1011·1001·1110으로 바꿔 이어 붙이면 $101110011110_{(2)}$이다.

계산 과정

$16 = 2^4$이므로 16진수 한 자리는 2진수 4자리와 대응한다. 따라서 **각 자리를 4비트로 바꾼 뒤 원래 순서대로 연결**하면 된다. 4비트의 자릿값은 왼쪽부터 8, 4, 2, 1이며, 필요한 자릿값에 1을 놓고 나머지에는 0을 놓는다.

B의 변환: 16진수 B는 10진수 11이다. 11 = 8 + 2 + 1이므로 자릿값 8, 2, 1에 해당하는 비트가 1이고, 4에 해당하는 비트는 0이다. 따라서 B는 1011로 바뀐다.

9의 변환: 9 = 8 + 1이므로 자릿값 8과 1에 해당하는 비트만 1이다. 따라서 9는 1001로 바뀐다.

E의 변환: 16진수 E는 10진수 14이다. 14 = 8 + 4 + 2이므로 자릿값 8, 4, 2에 해당하는 비트가 1이고, 1에 해당하는 비트는 0이다. 따라서 E는 1110으로 바뀐다.

B·9·E의 변환 결과를 왼쪽부터 연결하면 $B9E_{(16)} = 101110011110_{(2)}$이며, 이는 ③의 값과 일치한다.

NoteBase

문 14. 다음 프로그램의 구성 요소들 중 프로세스 내에서 생성한 스레드들 사이에 공유되지 않는 것을 모두 고르면?

ㄱ. 레지스터(Register)
ㄴ. 힙(Heap) 메모리
ㄷ. 전역 변수(Global variables)
ㄹ. 스택(Stack) 메모리

① ㄱ, ㄴ
② ㄱ, ㄹ
③ ㄴ, ㄷ
④ ㄷ, ㄹ

정답 2번

간단한 해설

스레드는 실행 위치와 함수 호출 상태를 독립적으로 유지해야 하므로 **레지스터 값과 스택을 스레드별로 따로 관리**한다.

NoteBase

보기별 해설

ㄱ. 레지스터: 공유되지 않는다. 레지스터에는 현재 실행 위치와 연산 중인 값 등 실행 상태가 담긴다. 스레드마다 실행 상태가 다르므로 **각 스레드의 레지스터 값을 별도로 보존**하며, 실행할 스레드가 바뀌면 그 스레드의 값을 복원한다.

ㄴ. 힙 메모리: 공유된다. 힙은 프로그램 실행 중 동적으로 할당한 데이터를 저장하는 영역이다. 같은 프로세스의 스레드들은 **동일한 주소 공간의 힙을 함께 사용**하므로, 한 스레드가 할당한 데이터에 다른 스레드도 해당 주소를 통해 접근할 수 있다.

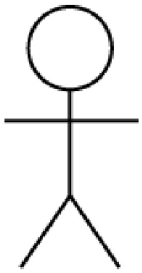
ㄷ. 전역 변수: 공유된다. 전역 변수는 특정 스레드의 함수 호출에 종속되지 않고 프로세스의 데이터 영역에 저장된다. 따라서 같은 전역 변수를 사용하는 스레드들은 **동일한 저장 위치**를 참조하며, 한 스레드가 변경한 값을 다른 스레드도 사용할 수 있다.

ㄹ. 스택 메모리: 공유되지 않는다. 스택에는 함수 호출에 따른 매개변수, 지역 변수, 복귀 주소 등이 저장된다. 스레드마다 함수 호출 흐름이 다르므로 **각자 별도의 스택**을 사용하여 호출 상태를 유지한다. 같은 함수를 실행하더라도 각 스레드의 지역 변수와 복귀 위치는 따로 관리된다.

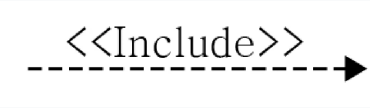
공유되지 않는 구성 요소는 ㄱ과 ㄹ이며, ②의 조합에 해당한다.

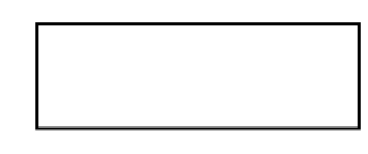
NoteBase

문 15. 다음 중 Use case diagram에서 사용하는 기본 요소가 아닌 것은?

① 

② 

③ 

④ 

정답 4번

간단한 해설

유스케이스에서 시스템이 제공하는 기능은 **타원**으로 나타낸다. ④의 사각형은 시스템의 범위를 나타내는 **선택적 경계 표시**로 사용할 수 있으나, 기본 요소인 액터·유스케이스·관계의 기호와는 구분한다.

NoteBase

도식 해설

유스케이스 다이어그램은 시스템이 제공하는 기능과 외부 참여자의 상호작용을 표현한다. 그림을 읽을 때는 참여자를 나타내는 기호, 기능을 나타내는 기호, 관계를 나타내는 화살표를 구분한다.

① 액터(Actor): 사람 모양의 기호는 시스템과 상호작용하는 **외부의 역할**을 나타낸다. 액터는 사용자뿐 아니라 외부 장치나 다른 시스템일 수도 있다.

② 확장 관계(Extend): 점선 화살표 위의 'Extend'는 기본 유스케이스에 **특정 조건에서 추가되는 동작**을 나타낸다. 화살표는 추가 동작을 정의한 확장 유스케이스에서 확장 대상인 기본 유스케이스로 향한다.

③ 포함 관계(Include): 점선 화살표 위의 'Include'는 한 유스케이스가 다른 유스케이스의 동작을 **자신의 동작 일부로 포함**하는 관계를 나타낸다. 화살표는 포함하는 유스케이스에서 포함되는 유스케이스로 향하며, 여러 유스케이스가 공통으로 사용하는 동작을 분리할 때 활용한다.

④ 사각형: 시스템이 제공하는 기능인 **유스케이스 자체는 타원**으로 표시하므로, 제시된 사각형을 유스케이스의 기호로 보아서는 안 된다. 다만 사각형으로 유스케이스들을 둘러싸면 시스템 내부와 외부를 구분하는 시스템 경계를 표현할 수 있다. 이 경계는 선택적인 보조 표시이므로, 이 문항의 기본 요소인 액터·유스케이스·관계와 구분한다.

NoteBase

문 16. 가상메모리(Virtual Memory)를 효과적으로 제공하기 위해 Core i7과 같은 프로세서 내부에 있는 장치는 무엇인가?

① TLB(Translation Lookaside Buffer)

② 캐시(Cache)

③ 페이지 테이블(Page Table)

④ 스왑 스페이스(Swap Space)

정답 1번

간단한 해설

가상메모리에서는 CPU가 사용하는 가상주소를 물리주소로 변환해야 한다. **TLB**는 최근 사용한 주소 변환 정보를 프로세서 내부에 저장하여 **페이지 테이블 조회를 줄이는** 장치이다.

공통 해설

가상메모리에서 가상주소는 프로그램이 사용하는 주소이고, 물리주소는 실제 주기억장치의 위치를 가리키는 주소이다. 페이징 방식에서는 페이지 테이블에 가상 페이지와 물리 프레임의 대응 정보를 저장한다. 주소를 변환할 때마다 페이지 테이블을 따라가면 시간이 걸리므로, 프로세서 내부의 TLB에 **최근 사용한 페이지의 주소 변환 정보**를 보관한다. 필요한 정보가 TLB에 있으면 페이지 테이블을 다시 조회하지 않고 물리주소를 구할 수 있다. 없으면 페이지 테이블을 조회하여 변환 정보를 얻고 TLB에 적재한다.

TLB도 캐시의 일종이지만, 이 문항에서 구별할 기준은 **명령어나 데이터 자체가 아니라 주소 변환 정보를 저장하는가**이다. 일반적인 명령어·데이터 캐시는 메모리의 내용을 빠르게 이용하기 위한 장치이고, TLB는 주소 변환을 빠르게 하기 위한 장치이다. 페이지 테이블은 주기억장치에 두는 주소 대응 자료구조이며, 스왑 스페이스는 메모리에서 내보낸 페이지 등을 보관하는 보조기억장치 영역이다. 따라서 프로세서 내부에서 가상메모리의 주소 변환을 가속하는 장치는 TLB에 해당한다.

NoteBase

문 17. 다음 중 C 프로그래밍 언어의 식별자로 사용할 수 없는 것은?

① 3id

② My_ID

③ __yes

④ K

정답 1번

간단한 해설

C 식별자는 **숫자로 시작할 수 없다**. '3id'는 첫 글자가 숫자 3이므로 식별자의 문자 구성 규칙을 만족하지 못한다.

공통 해설

식별자는 변수나 함수 등을 구별하기 위해 붙이는 이름이다. 제시된 영문자·숫자·밑줄 조합에서는 **첫 글자가 영문자 또는 밑줄(_)**이어야 하며, 그 뒤에는 영문자·숫자·밑줄을 이어 쓸 수 있다. 따라서 숫자가 맨 앞에 있는 '3id'는 식별자가 될 수 없지만, 숫자를 뒤로 옮긴 'id3'는 문자 구성 규칙에 맞는다.

'My_ID', '__yes', 'K'는 모두 이 문자 구성 규칙을 만족한다. 밑줄은 중간이나 처음에 올 수 있고, 영문자 한 글자만으로도 식별자를 구성할 수 있다. 이 문항에서 '3id'를 구별하는 기준은 숫자로 시작하는지 여부이다.

다만 **문자 구성 규칙을 만족하는 것과 사용자가 자유롭게 이름으로 쓸 수 있는 것은 구별**해야 한다. C 표준에서 '__yes'처럼 밑줄 두 개로 시작하는 식별자는 구현용으로 예약되어 있으므로, 사용자가 변수나 함수의 이름으로 정의해서는 안 된다.

NoteBase

문 18. 3개의 page를 수용할 수 있는 메모리가 있으며, 현재 완전히 비어 있다. 어느 프로그램이 <보기>와 같이 page 번호를 요청했을 때, LRU(Least-Recently-Used)를 사용할 경우 몇 번의 page-fault가 발생하는가?

<보기>
요청하는 번호순서 : 2 3 2 1 5 2 4 5

① 6번
② 5번
③ 4번
④ 3번

정답 2번

간단한 해설

LRU는 메모리가 가득 차면 **마지막으로 사용한 지 가장 오래된 페이지**를 교체한다. 페이지가 이미 메모리에 있어도 사용 시점은 갱신되므로, 주어진 요청에서 페이지 부재는 **총 5번** 발생한다.

NoteBase

계산 과정

페이지 부재(page-fault)는 요청한 페이지가 메모리에 없을 때 발생한다. 빈 공간이 있으면 그곳에 적재하고, 3칸이 모두 차 있으면 LRU에 따라 마지막 사용 시점이 가장 오래된 페이지를 내보낸다. 아래 대괄호에는 메모리에 있는 페이지를 오래전에 사용한 것부터 최근에 사용한 것 순으로 표시한다.

1번째 요청 2: 메모리가 비어 있으므로 페이지 부재가 발생하고 2를 적재한다. 사용 순서는 [2]이며, 누적 부재 횟수는 1번이다.

2번째 요청 3: 메모리에 3이 없으므로 페이지 부재가 발생한다. 빈 공간에 3을 적재하면 [2, 3]이 되고, 누적 부재 횟수는 2번이다.

3번째 요청 2: 2가 이미 있으므로 페이지 부재는 발생하지 않는다. 다만 **2의 마지막 사용 시점이 갱신**되어 사용 순서는 [3, 2]가 된다.

4번째 요청 1: 메모리에 1이 없으므로 페이지 부재가 발생한다. 남은 빈 공간에 1을 적재하면 [3, 2, 1]이 되고, 누적 부재 횟수는 3번이다.

5번째 요청 5: 메모리에 5가 없고 3칸이 모두 차 있으므로 페이지 부재와 교체가 발생한다. 마지막 사용 시점은 3이 2번째 요청, 2가 3번째 요청, 1이 4번째 요청이므로 **가장 오래전에 사용한 3을 교체**한다. 사용 순서는 [2, 1, 5]가 되고, 누적 부재 횟수는 4번이다.

6번째 요청 2: 2가 이미 있으므로 페이지 부재는 발생하지 않는다. 2를 가장 최근에 사용했으므로 사용 순서는 [1, 5, 2]로 바뀐다.

7번째 요청 4: 메모리에 4가 없으므로 페이지 부재가 발생한다. 현재 가장 오래전에 사용한 페이지는 1이므로 **1을 내보내고 4를 적재**한다. 사용 순서는 [5, 2, 4]가 되고, 누적 부재 횟수는 5번이다.

8번째 요청 5: 5가 메모리에 남아 있으므로 페이지 부재는 발생하지 않는다. 사용 순서는 [2, 4, 5]가 된다.

페이지 부재는 1·2·4·5·7번째 요청에서 발생하므로 총 5번이며, 선택지 ②의 값과 일치한다.

NoteBase

문 19. 소프트웨어 프로젝트 관리가 어려운 이유로 옳지 않은 것은?

① 소프트웨어는 형태가 없어 프로젝트 관리자는 프로젝트 진척사항을 분석하는 데 어려움이 있다.

② 소프트웨어 개발 프로세스는 조직에 따라 가변적이므로 관리에 어려움이 있다.

③ 컴퓨터와 통신에서의 빠른 기술적 변화로 인해 관리자의 경험이 새로운 프로젝트에 전달되지 않을 수 있다.

④ 대규모 소프트웨어 프로젝트는 일회성(one-off) 프로젝트가 전혀 없어서, 경험이 충분한 관리자가 문제를 예측할 수 없다.

정답 4번

간단한 해설

④의 '일회성 프로젝트가 전혀 없다'는 설명이 잘못되었다. 대규모 소프트웨어 프로젝트는 요구사항과 개발 환경이 서로 다른 **일회성(one-off) 프로젝트인 경우가 많아**, 경험이 풍부한 관리자도 새로운 문제를 예측하기 어렵다.

선지별 해설

① O: 소프트웨어는 건물처럼 외형을 보고 완성 정도를 확인할 수 없는 **무형의 산출물**이다. 따라서 겉으로 드러나는 모습만으로 진척사항을 파악하기 어렵고, 개발 문서나 구현·시험 결과 등을 확인해야 한다.

② O: 소프트웨어 개발 프로세스는 개발에 필요한 활동과 그 수행 방식이다. 조직마다 인력, 업무 방식, 개발 대상 등이 달라 **프로세스도 조직별로 달라질 수 있다**. 따라서 다른 조직에서 사용한 관리 방식이나 경험을 그대로 적용하여 진행 상황과 문제 발생을 예측하기 어렵다.

③ O: 컴퓨터와 통신 기술이 빠르게 변하면 사용하는 기술과 개발 환경도 달라진다. 이 때문에 **과거 경험이 새로운 환경에 그대로 적용되지 않을 수 있어**, 이전 프로젝트에서 얻은 교훈만으로 새로운 프로젝트의 문제를 예측하기 어렵다.

④ X: 대규모 소프트웨어 프로젝트에는 이전 프로젝트와 요구사항이나 환경이 다른 **일회성(one-off) 프로젝트가 많다**. 여기서 일회성이란 동일한 조건의 작업을 반복하는 것이 아니라, 해당 프로젝트만의 특수성이 크다는 뜻이다. 따라서 '일회성 프로젝트가 전혀 없어서'를 '일회성 프로젝트인 경우가 많아서'로 고쳐야 한다. 이러한 차이 때문에 경험이 풍부한 관리자도 문제를 충분히 예측하기 어려울 수 있다.

NoteBase

문 20. 다음 C 프로그램의 실행 결과는?

```
#include<stdio.h>
struct student
{
    char name[20];      // 이름
    int money;          // 돈
    struct student* link; // 자기 참조 구조체 포인터 변수
};
int main(void)
{
    struct student stu1 = {"Kim", 90, NULL};
    struct student stu2 = {"Lee", 80, NULL};
    struct student stu3 = {"Goo", 60, NULL};
    stu1.link = &stu2;
    stu2.link = &stu3;

    printf("%s %d ₩n", stu1.link->link->name, stu1.link->money);
    return 0;
}
```

① Goo 80

② Lee 60

③ Goo 60

④ Lee 80

정답 1번

간단한 해설

이름은 링크를 두 번 따라간 **stu3의 name**을, 돈은 링크를 한 번 따라간 **stu2의 money**를 출력한다. 따라서 이름은 Goo, 돈은 80이다.

NoteBase

계산 과정

링크 설정: stu1, stu2, stu3의 이름과 돈은 각각 Kim·90, Lee·80, Goo·60으로 초기화된다. stu1.link = &stu2는 stu1의 link에 stu2의 주소를 저장하고, stu2.link = &stu3는 stu2의 link에 stu3의 주소를 저장한다. 따라서 연결 관계는 stu1 → stu2 → stu3이며, stu3.link는 초기값 NULL을 유지한다.

이름 접근: 구조체 변수의 멤버는 .으로, 구조체 포인터가 가리키는 대상의 멤버는 ->로 접근한다. stu1.link는 stu2를 가리키고, stu1.link->link는 stu2의 link이므로 stu3를 가리킨다. 여기에 ->name을 적용한 stu1.link->link->name은 **stu3의 이름 Goo**를 나타낸다.

돈 접근: stu1.link->money는 stu1.link가 가리키는 stu2의 money에 접근하므로 값이 80이다. 이름을 구하는 식과 달리 **stu2에서 link를 한 번 더 따라가지 않는다**. 따라서 stu3의 money인 60은 출력에 사용되지 않는다.

출력 결과: printf의 %s에는 문자열 Goo가, %d에는 정수 80이 대응한다. 따라서 출력되는 이름과 돈은 'Goo 80'이며, ①의 실행 결과와 일치한다.

NoteBase