



시험에 나오는것만 공부한다!

시나공시리즈

기출문제

2025년 2회 정보처리기사 실기



정보처리기사 실기 시험은 한국산업인력공단에서 문제를 공개하지 않아 문제 복원에 많은 어려움이 있습니다. 다음에 제시된 문제는 시험을 치른 학생들의 기억을 토대로 복원한 것이므로, 일부 내용이나 문제별 배점이 실제 시험과 다를 수 있음을 알립니다.

저작권 안내

이 자료는 시나공 카페 회원을 대상으로 하는 자료로서 개인적인 용도로만 사용할 수 있습니다. 허락 없이 복제하거나 다른 매체에 옮겨 실을 수 없으며, 상업적 용도로 사용할 수 없습니다.

*** 수험자 유의사항 ***

1. 시험 문제지를 받는 즉시 응시하고자 하는 종목의 문제지가 맞는지를 확인하여야 합니다.
2. 시험 문제지 총면수·문제번호 순서·인쇄상태 등을 확인하고, 수험번호 및 성명을 답안지에 기재하여야 합니다.
3. 문제 및 답안(지), 채점기준은 일절 공개하지 않으며 자신이 작성한 답안, 문제 내용 등을 수험표 등에 이기(옮겨 적는 행위) 등은 관련 법 등에 의거 불이익 조치 될 수 있으니 유의하시기 바랍니다.
4. 수험자 인적사항 및 답안작성(계산식 포함)은 흑색 기구만 사용하되, 동일한 한 가지 색의 필기구만 사용해야 하며 흑색을 제외한 유색 필기구 또는 연필류를 사용하거나 2가지 이상의 색을 혼합 사용하였을 경우 그 문항은 0점 처리됩니다.
5. 답란(답안 기재란)에는 문제와 관련 없는 불필요한 낙서나 특이한 기록사항 등을 기재하여서는 안되며 부정의 목적으로 특이한 표식을 하였다고 판단될 경우에는 모든 문항이 0점 처리됩니다.
6. 답안을 정정할 때에는 반드시 정정부분을 두 줄(=)로 그어 표시하여야 하며, 두 줄로 굿지 않은 답안은 정정하지 않은 것으로 간주합니다.
7. 답안의 한글 또는 영문의 오타자는 오답으로 처리됩니다. 단, 답안에서 영문의 대·소문자 구분, 띄어쓰기는 여부에 관계 없이 채점합니다.
8. 계산 또는 디버깅 등 계산 연습이 필요한 경우는 <문 제> 아래의 연습란을 사용하시기 바라며, 연습란은 채점대상이 아닙니다.
9. 문제에서 요구한 가지 수(항수) 이상을 답란에 표기한 경우에는 답안기재 순으로 요구한 가지 수(항수)만 채점하고 한 항에 여러 가지를 기재하더라도 한 가지로 보며 그 중 정답과 오답이 함께 기재란에 있을 경우 오답으로 처리됩니다.
10. 한 문제에서 소문제로 파생되는 문제나, 가지수를 요구하는 문제는 대부분의 경우 부분채점을 적용합니다. 그러나 소문제로 파생되는 문제 내에서의 부분 배점은 적용하지 않습니다.
11. 답안은 문제의 마지막에 있는 답란에 작성하여야 합니다.
12. 부정 또는 불공정한 방법(시험문제 내용과 관련된 메모지사용 등)으로 시험을 치른 자는 부정행위자로 처리되어 당해 시험을 중지 또는 무효로 하고, 2년간 국가기술자격검정의 응시자격이 정지됩니다.
13. 시험위원이 시험 중 신분확인을 위하여 신분증과 수험표를 요구할 경우 반드시 제시하여야 합니다.
14. 시험 중에는 통신기기 및 전자기기(휴대용 전화기 등)를 지참하거나 사용할 수 없습니다.
15. 국가기술자격 시험문제는 일부 또는 전부가 저작권법상 보호되는 저작물이고, 저작권자는 한국산업인력공단입니다. 문제의 일부 또는 전부를 무단 복제, 배포, 출판, 전자출판 하는 등 저작권을 침해하는 일체의 행위를 금합니다.

※ 수험자 유의사항 미준수로 인한 채점상의 불이익은 수험자 본인에게 전적으로 책임이 있음

문제 1 다음 설명에서 괄호에 공통으로 들어갈 알맞은 답을 쓰시오. (5점)



파일의 구조는 파일을 구성하는 레코드들이 보조기억장치에 편성되는 방식을 의미하는 것으로, 크게 순차, (), 해싱으로 구분한다. () 파일 구조는 <값, 주소> 쌍으로 구성되는 데이터 구조를 활용하여 데이터에 접근하는 방식으로, 자기 디스크에서 주로 활용된다.

답 :

문제 2 다음 설명에 해당하는 알맞은 용어를 영문 3글자로 쓰시오. (5점)



- 다른 컴퓨터에 로그인, 원격 명령 실행, 파일 복사 등을 수행할 수 있도록 다양한 기능을 지원하는 프로토콜 또는 이를 이용한 응용 프로그램이다.
- 데이터 암호화와 강력한 인증 방법으로 보안성이 낮은 네트워크에서도 안전하게 통신할 수 있다.
- 키(key)를 통한 인증 방법을 사용하려면 사전에 클라이언트의 공개키를 서버에 등록해야 한다.
- 기본적으로는 22번 포트를 사용한다.

답 :

문제 3 다음 설명에 해당하는 용어를 <보기>에서 찾아 기호(㉠~㉤)로 쓰시오. (5점)



- 데이터베이스를 구성하는 가장 작은 논리적 단위로, 릴레이션에서 열(Column)에 해당한다.
- 파일 구조상의 데이터 항목 또는 데이터 필드에 해당된다.
- 개체의 특성을 기술한다.

<보기>

- | | | | |
|----------|----------------|---------------|-------------|
| ㉠ Domain | ㉡ Tuple | ㉢ Attribute | ㉣ Integrity |
| ㉤ Degree | ㉥ Relationship | ㉦ Cardinality | ㉧ Entity |

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 4 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
public class Main {
    public static void change(String[] data, String s){
        data[0] = s;
        s = "Z";
    }
    public static void main(String[] args) {
        String data[] = { "A" };
        String s = "B";
        change(data, s);
        System.out.print(data[0] + s);
    }
}
```

답 :



문제 5 다음 C 언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
    int p;
    struct node* n;
};
int main( ) {
    struct node a = {1, NULL};
    struct node b = {2, NULL};
    struct node c = {3, NULL};
    a.n = &b; b.n = &c; c.n = NULL;
    c.n = &a; a.n = &b; b.n = NULL;
    struct node* head = &c;
    printf("%d %d %d", head -> p, head -> n -> p, head -> n -> n -> p);
    return 0;
}
```

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.

문제 6 스케줄링에 대한 다음 설명에서 괄호(①, ②)에 들어갈 알맞은 용어를 쓰시오. (5점)



- (①)는 준비상태 큐에서 기다리고 있는 프로세스들 중에서 실행 시간이 가장 짧은 프로세스에게 먼저 CPU를 할당하는 기법이다. 가장 적은 평균 대기 시간을 제공하는 최적 알고리즘이지만, 실행 시간이 긴 프로세스는 실행 시간이 짧은 프로세스에게 할당 순위가 밀려 무한 연기 상태가 발생할 수 있다.
- (②)는 현재 실행중인 프로세스의 남은 시간과 준비상태 큐에 새로 도착한 프로세스의 실행 시간을 비교하여 가장 짧은 실행 시간을 요구하는 프로세스에게 CPU를 할당하는 기법으로, 시분할 시스템에 유용하다. 준비상태 큐에 있는 각 프로세스의 실행 시간을 추적하여 보유하고 있어야 하므로 오버헤드가 증가한다.

답

- ①
- ②

문제 7 다음은 <EMPLOYEE> 릴레이션에 대해 <관계 대수식>을 수행했을 때 출력되는 <결과>이다. <결과>의 각 괄호(①~⑤)에 들어갈 알맞은 답을 쓰시오. (5점)



<관계 대수식>

$\pi_{TTL}(EMPLOYEE)$

<EMPLOYEE>

INDEX	AGE	TTL
1	48	부장
2	25	대리
3	41	과장
4	36	차장



<결과>

①
②
③
④
⑤

답

- ①
- ②
- ③
- ④
- ⑤

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 8 다음 IP 주소와 서브넷 마스크에 대한 내용에서 괄호(①, ②)에 들어갈 알맞은 답을 쓰시오. (5점)

호스트의 IP 주소가 223.13.234.132이고, 서브넷 마스크가 255.255.255.192일 때, 호스트가 속한 네트워크 주소는 223.13.234.(①)이고, 해당 네트워크에서 사용할 수 있는 호스트 수는 (②)개이다.

답 :



문제 9 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
public class Main {
    static interface F {
        int apply(int x) throws Exception;
    }
    public static int run(F f) {
        try {
            return f.apply(3);
        } catch (Exception e) {
            return 7;
        }
    }

    public static void main(String[] args) {
        F f = (x) -> {
            if (x > 2) {
                throw new Exception( );
            }
            return x * 2;
        };
        System.out.print(run(f) + run((int n) -> n + 9));
    }
}
```

답 :

※ 다음 여백은 연습란으로 사용하시기 바랍니다.

연 습 란



문제 10 다음 C 언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
#include <stdio.h>
#define SIZE 3

typedef struct {
    int a[SIZE];
    int front;
    int rear;
} Queue;

void enq(Queue* q, int val) {
    q -> a[q -> rear] = val;
    q -> rear = (q -> rear + 1) % SIZE;
}

int deq(Queue* q) {
    int val = q -> a[q -> front];
    q -> front = (q -> front + 1) % SIZE;
    return val;
}

int main( ) {
    Queue q = {{0}, 0, 0};
    enq(&q, 1);
    enq(&q, 2);
    deq(&q);
    enq(&q, 3);
    printf("%d 그리고 %d", deq(&q), deq(&q));
    return 0;
}
```

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 11 클라이언트와 서버 간 자바 스크립트 및 XML을 비동기 방식으로 처리하며, 전체 페이지를 새로 고치지 않고도 웹 페이지 일부 영역만을 업데이트할 수 있도록 하는 기술을 의미하는 용어를 쓰시오. (5점)

답 :



문제 12 다음 설명에서 괄호에 들어갈 알맞은 디자인 패턴을 아래에서 찾아 쓰시오. (5점)

생성 패턴	구조 패턴	행위 패턴
Abstract Factory	Adapter	Chain of Responsibility
Builder	Bridge	Command
Factory Method	Composite	Interpreter
Prototype	Decorator	Iterator
Singleton	Facade	Mediator
	Proxy	Memento
		Observer
		Strategy
		Template Method

()은/는 복잡한 시스템을 개발하기 쉽도록 클래스나 객체들을 조합하는 패턴에 속하며, 대리자라는 이름으로도 불린다. 내부에서는 객체 간의 복잡한 관계를 단순하게 정리해 주고, 외부에서는 객체의 세부적인 내용을 숨기는 역할을 한다.

답 :



문제 13 서비스 거부(DoS) 공격 기법 중 TCP가 신뢰성 있는 전송을 위해 사용하는 3-way-handshake 과정을 의도적으로 중단시킴으로써 공격 대상지인 서버가 대기 상태에 놓여 정상적인 서비스를 수행하지 못하게 하는 공격 방법을 가리키는 용어를 쓰시오. (5점)

답 :

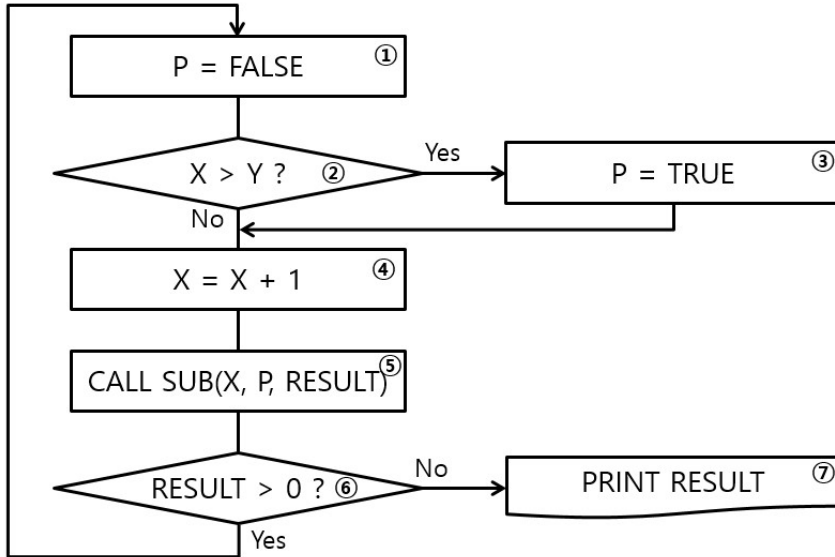
※ 다음 여백은 연습란으로 사용하시기 바랍니다.

연 습 란



문제 14 다음은 화이트박스 테스트의 프로그램 제어흐름이다. 다음의 순서도를 참고하여 분기 커버리지로 구성할 테스트 케이스를 작성하시오. (5점)

<순서도>



<작성예시>

(1) → (2) → (4)

답 :

() → () → () → () → () → () → ()
 () → () → () → () → () → ()



문제 15 준비상태 큐에 각 프로세스의 도착 시간과 실행 시간이 다음과 같을 때 RR(Round Robin)로 스케줄링 할 때 평균 대기 시간을 쓰시오. (단, Time Slice는 4초이다.) (5점)

프로세스	도착 시간	실행 시간
A	0	8
B	1	4
C	2	9
D	3	5

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 16 다음 C 언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
#include <stdio.h>

struct dat {
    int x;
    int y;
};

int main( ) {
    struct dat a[] = {{1, 2}, {3, 4}, {5, 6}};
    struct dat* ptr = a;
    struct dat** pptr = &ptr;
    (*pptr)[1] = (*pptr)[2];
    printf("%d 그리고 %d", a[1].x, a[1].y);
    return 0;
}
```

답 :



문제 17 다음 Python으로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
lst = [1,2,3]
dst = {i : i* 2 for i in lst}
s = set(dst.values( ))
lst[0] = 99
dst[2] = 7
s.add(99)
print(len(s & set(dst.values( ))))
```

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 18 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
public class Main {
    public static class Parent {
        public int x(int i) { return i + 2; }
        public static String id( ) { return "P"; }
    }
    public static class Child extends Parent {
        public int x(int i) { return i + 3; }
        public String x(String s) { return s + "R"; }
        public static String id( ) { return "C"; }
    }
    public static void main(String[] args) {
        Parent ref = new Child( );
        System.out.println(ref.x(2) + ref.id( ));
    }
}
```

답 :

시나공

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 19 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
public class Main {
    public static class B0 {
        public int v;
        public B0(int v) {
            this.v = v;
        }
    }
    public static void main(String[] args) {
        B0 a = new B0(1);
        B0 b = new B0(2);
        B0 c = new B0(3);
        B0[] arr = {a, b, c};
        B0 t = arr[0];
        arr[0] = arr[2];
        arr[2] = t;
        arr[1].v = arr[0].v;
        System.out.println(a.v + "a" + b.v + "b" + c.v);
    }
}
```

답 :

시나공

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.



문제 20 다음 C 언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.) (5점)

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    char c;
    struct node* p;
};

struct node* func(char* s) {
    struct node* h = NULL, *n;
    while(*s) {
        n = malloc(sizeof(struct node));
        n -> c = *s++;
        n -> p = h;
        h = n;
    }
    return h;
}

int main( ) {
    struct node* n = func("BEST");
    while(n) {
        putchar(n -> c);
        struct node* t = n;
        n = n -> p;
        free(t);
    }
    return 0;
}
```

답 :

연 습 란

※ 다음 여백은 연습란으로 사용하시기 바랍니다.

기출문제 정답 및 해설

[문제 1]

※ 다음 중 하나를 쓰면 됩니다.

색인, Index

[문제 2]

SSH

[문제 3]

㉠

[문제 4]

BB

[해설]

```
public class Main {
    ④    public static void change(String[] data, String s){
    ⑤        data[0] = s;
    ⑥        s = "Z";
    }
    public static void main(String[] args) {
    ①        String data[] = { "A" };
    ②        String s = "B";
    ③        change(data, s);
    ⑦        System.out.print(data[0] + s);
    }
}
```

모든 Java 프로그램은 반드시 main() 메소드에서 시작한다.

① 문자열 배열 data를 선언하고 “A”로 초기화한다.

※ 다음 그림에서 “A”가 저장된 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했습니다.

주소	메모리			
	1Byte			
1000	'A'			
	data[0]			

② 문자열 변수 s를 선언하고 “B”로 초기화한다.

③ data와 s를 인수로 change() 메소드를 호출한다.

※ 인수로 배열의 이름을 지정하면 배열의 시작 주소가 인수로 전달됩니다.

④ 반환값이 없는 change() 메소드의 시작점이다. ③번에서 전달한 data 배열의 주소와 s를 data와 s가 받는다.

⑤ data[0], 즉 data 배열의 첫 번째 값에 s의 값 “B”를 저장한다.

주소	메모리			
	1Byte			
1000	'B'			
	data[0]			

⑥ s에 “Z”를 저장한다. 메소드가 종료되었으므로 메소드를 호출했던 ③번의 다음 줄인 ⑦번으로 이동한다.

※ change() 메소드의 문자열 변수 s는 지역 변수로, change() 메소드 안에서만 사용되므로 main() 메소드의 문자열 변수 s에는 영향을 주지 않습니다.

⑦ data[0]의 값 “B”에 s의 값 “B”를 덧붙여 출력한다.

결과 BB

[문제 5]

3 1 2

[해설]

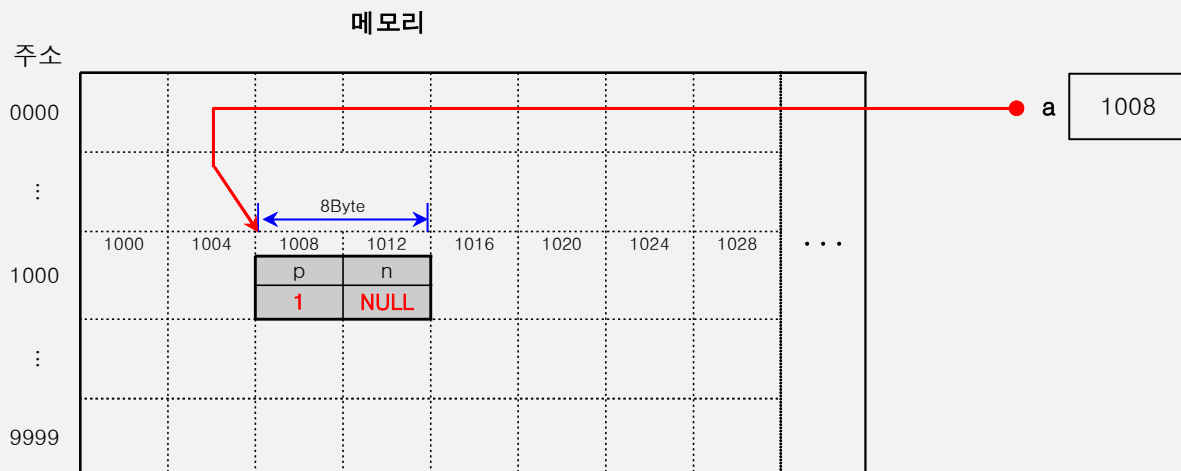
```
#include <stdio.h>
struct node {                // node 구조체를 정의한다.
    int p;                   // 정수형 변수 p를 선언한다.
    struct node *n;          // node 구조체의 포인터 변수 n을 선언한다.
};
```

struct node	
int p (4Byte)	struct node *n (4Byte)
데이터를 저장할 멤버	다음 노드의 주소를 저장할 멤버

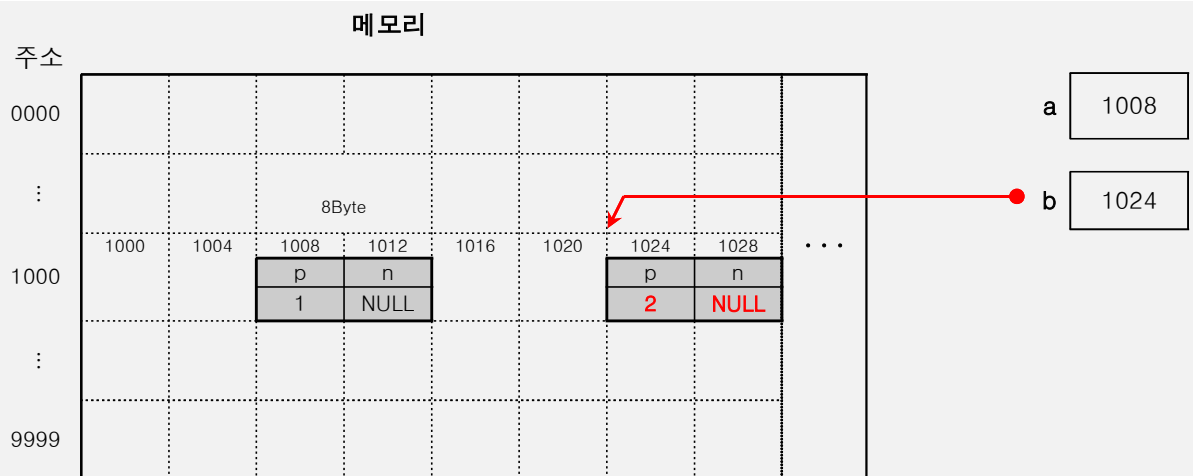
```
int main( ) {
    ① struct node a = {1, NULL};
    ② struct node b = {2, NULL};
    ③ struct node c = {3, NULL};
    ④ a.n = &b; b.n = &c; c.n = NULL;
    ⑤ c.n = &a; a.n = &b; b.n = NULL;
    ⑥ struct node* head = &c;
    ⑦ printf("%d %d %d", head -> p, head -> n -> p, head -> n -> n -> p);
    ⑧ return 0;
}
```

모든 C 언어 프로그램은 반드시 main() 함수에서 시작한다.

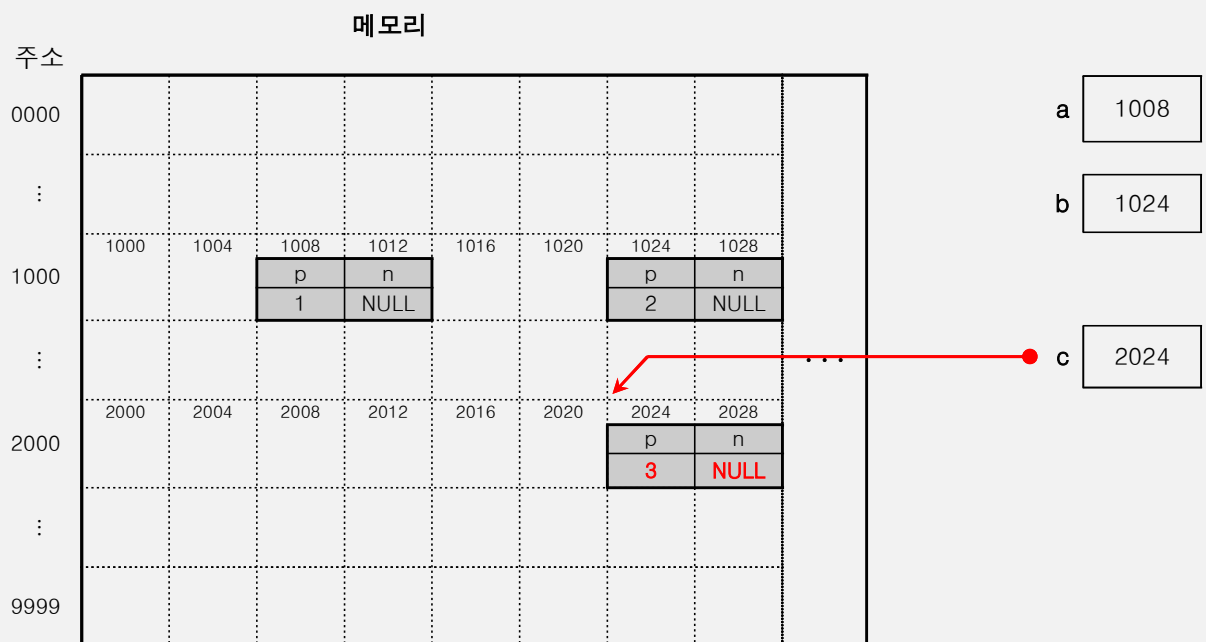
- ① node 자료형 변수 a를 선언하고, node의 p에 1을 node의 n에 NULL을 저장한다. a가 가리키고 있는 1008번지는 node 구조체의 크기만큼 할당된 공간이므로 a는 1008번지 이후의 8Byte를 의미한다.
(이후 그림에서 지정한 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했음)
※ NULL은 값이 없음을 의미하며, n에 NULL이 저장되어 있는 것은 어떤 곳도 가리키지 않고 있음을 의미합니다.



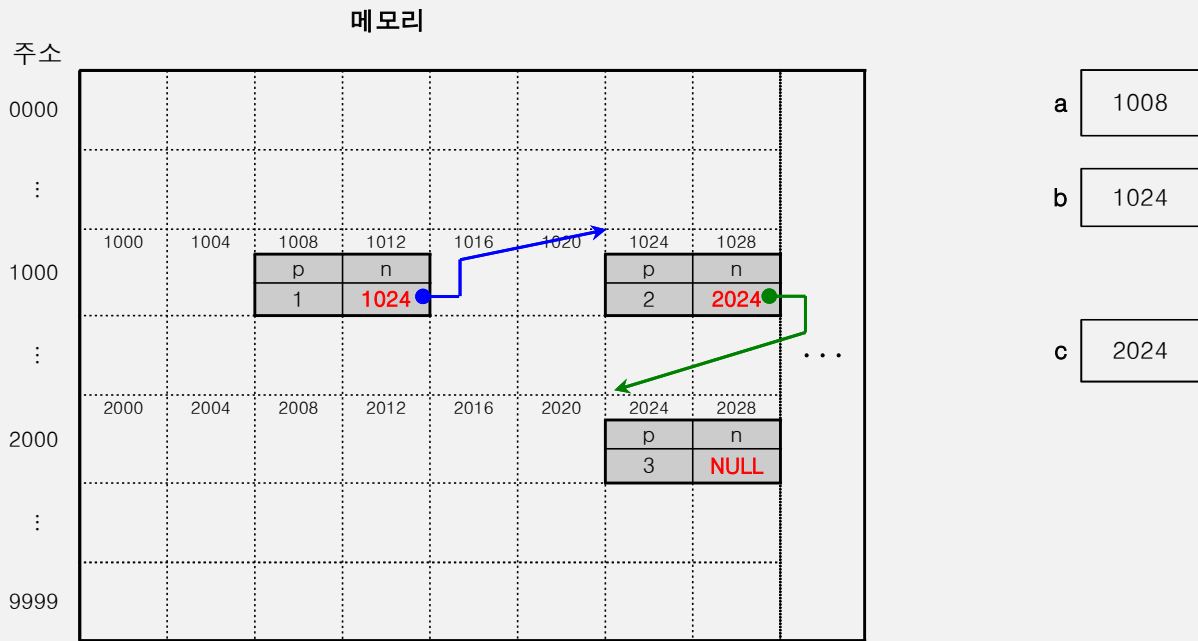
- ② node 자료형 변수 b를 선언하고, 2와 NULL을 저장한다.



③ node 자료형 변수 c를 선언하고, 3과 NULL을 저장한다.



- ④ a의 n에 b의 주소를, b의 n에 c의 주소를, c의 n에 NULL을 저장한다. n은 다음 노드의 주소를 가리키는 것이므로, a의 다음 노드는 b이고, b의 다음 노드는 c이다. 즉, 노드의 순서는 'a → b → c'가 된다.



[문제 6]

※ 문항별로 다음 중 하나를 쓰면 됩니다.

- ① SJF, Shortest Job First
- ② SRT, Shortest Remaining Time

[문제 7]

- ① TTL ② 부장 ③ 대리 ④ 과장 ⑤ 차장

[해설]

문제의 <관계 대수식>에서 사용된 π 는 주어진 릴레이션에서 속성 리스트(Attribute List)에 제시된 속성 값만을 추출하여 새로운 릴레이션을 만드는 PROJECT 연산이므로, <EMPLOYEE> 릴레이션에서 'TTL' 속성이 추출되어 속성명인 'TTL'부터 모든 속성값이 <결과>로 나타납니다.

[문제 8]

- ① 128 ② 62

[문제 9]

19

[해설]

```
public class Main {
    ① static interface F {
        int apply(int x) throws Exception;
    }
    ② public static int run(F f) {
    ③ try {
    ④ return f.apply(3);
    ⑤ } catch (Exception e) {
    ⑥ return 7;
    ⑦ }
    }

    public static void main(String[] args) {
    ⑧ F f = (x) -> {
        if (x > 2) {
            throw new Exception( );
        }
        return x * 2;
    };
    ⑨ System.out.print(run(f) + run((int n) -> n + 9));
    }
}
```

① 함수형 인터페이스 F를 정의한다. F는 정수를 받아 정수를 반환하고, 예외(Exception)를 발생시킬 수 있다.

② 인터페이스 F 타입의 객체 f를 선언한다. f는 인터페이스 F에 속한 apply(int x) 메소드가 호출되었을 때, x가 2보다 크면 예외를 발생시키고 그렇지 않으면 x * 2를 반환한다.

※ Java의 람다식

- 형식 : 인수 -> 실행코드
- 메소드 이름, 반환 타입, 인수 타입 없이 간단하게 미리 선언된 함수형 인터페이스를 구현할 수 있는 것으로, 코드를 더 짧고 읽기 쉽게 만들어주는 형식입니다.

모든 Java 프로그램은 반드시 main() 메소드에서 시작한다.

- ❶ 인터페이스 F의 객체 f를 인수로 run() 메소드를 호출한 후 반환받은 값과 정수 n을 인수로 run() 메소드를 호출한 후 돌려받은 값을 더한 값을 출력한다. 먼저 객체 f를 인수로 run() 메소드를 호출한다. ❷번으로 이동한다.
- ❷ 정수를 반환하는 run() 메소드의 시작점이다. ❶번에서 전달한 객체 f를 f가 받는다.
- ❸ 예외 구문의 시작이다.
- ❹ 3을 인수로 f의 apply() 메소드를 호출한다. ❶번에서 전달한 f에 ⑧ 람다식이 적용되었으므로, ❺번으로 이동한다.
- ❺ ❹번에서 전달한 값 3은 2보다 크므로 예외를 발생시킨다. 예외가 발생되었으므로 try 블록이 종료되고 catch 블록으로 제어가 이동된다. ❻번으로 이동한다.
※ throw : 강제로 예외를 발생시키는 키워드
- ❻ 예외가 발생한 경우 예외를 다루는 catch 문의 시작이다.
- ❼ 7을 메소드를 호출했던 ❸번으로 반환한다.
- ❽ 이어서 정수 n을 인수로 run() 메소드를 호출한다. ❾번으로 이동한다.
- ❾ 정수를 반환하는 run() 메소드의 시작점이다.
- ❿ 예외 구문의 시작이다.
- ⓫ 3을 인수로 f의 apply() 메소드를 호출한다. '(int n) -> n + 9', 즉 메소드에 전달되는 인수에 9를 더하는 람다식이 적용된 결과인 12를 메소드를 호출했던 ❿번으로 반환한다.
- ⓫ ❷번에서 반환받은 7과 ⓫번에서 반환받은 12를 더한 값 19를 출력한다.

결과 19

[문제 10]

3 그리고 2

[해설]

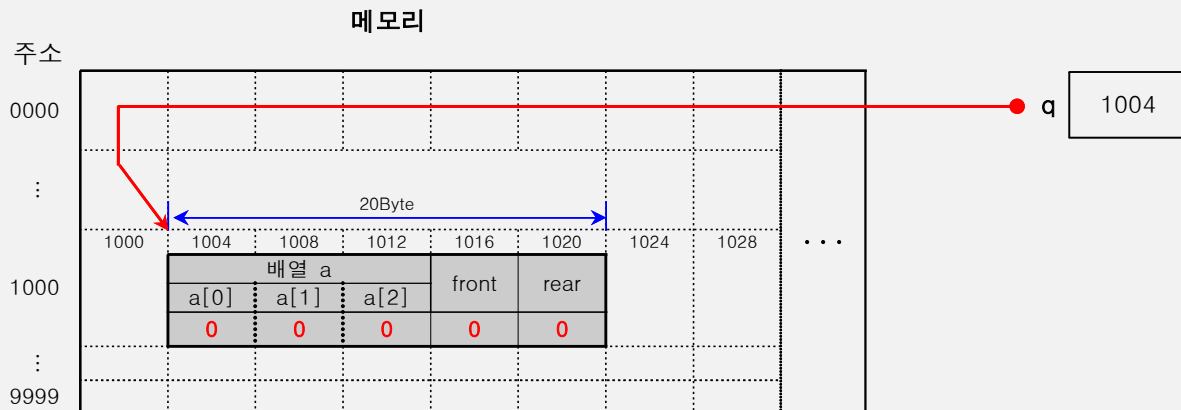
```
#include <stdio.h>
#define SIZE 3 // 3을 SIZE로 정의한다. 프로그램 안에서 SIZE는 숫자 3과 동일하다.

typedef struct { // 구조체를 정의한다.
    int a[SIZE]; // 크기가 3인 정수형 배열 a를 선언한다.
    int front; // 정수형 변수 front를 선언한다.
    int rear; // 정수형 변수 rear을 선언한다.
} Queue; // 구조체의 이름은 Queue이다.
```

```
int main( ) {
    ❶ Queue q = {{0}, 0, 0};
    ❷ enq(&q, 1);
    enq(&q, 2);
    deq(&q);
    enq(&q, 3);
    printf("%d 그리고 %d", deq(&q), deq(&q));
    return 0;
}
```

모든 C 언어 프로그램은 반드시 main() 함수에서 시작한다.

- ❶ Queue 자료형 변수 `q`를 선언하고, Queue의 배열 `a`, `front`, `rear`을 각각 0으로 초기화한다. `q`가 가리키고 있는 1004 번지는 Queue 구조체의 크기만큼 할당된 공간이므로 `q`는 1004번지 이후의 20Byte를 의미한다. (이후 그림에서 지정한 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했다.)



- ❷ `q`의 주소와 1을 인수로 `enq()` 함수를 호출한다.

```
❸ void enq(Queue* q, int val) {
❹   q -> a[q -> rear] = val;
❺   q -> rear = (q -> rear + 1) % SIZE;
}
```

- ❸ 반환값이 없는 `enq()` 함수의 시작점이다. ❷번에서 전달한 배열 `a`의 주소를 `q`가, 1을 `val`이 받는다.
 ❹ `q`가 가리키는 `rear`은 0이므로 `q`가 가리키는 `a[0]`에 `val`의 값 1을 저장한다.
 ❺ `q`가 가리키는 `rear`에 '`(q -> rear + 1) % SIZE`', 즉 `(0 + 1) % 3`이므로 1을 저장한 후 제어를 `enq()` 함수를 호출했던 ❷번의 다음 줄인 ❻번으로 옮긴다.



```
int main( ) {
❶ Queue q = {{0}, 0, 0};
❷ enq(&q, 1);
❸ enq(&q, 2);
  deq(&q);
  enq(&q, 3);
  printf("%d 그리고 %d", deq(&q), deq(&q));
  return 0;
}
```

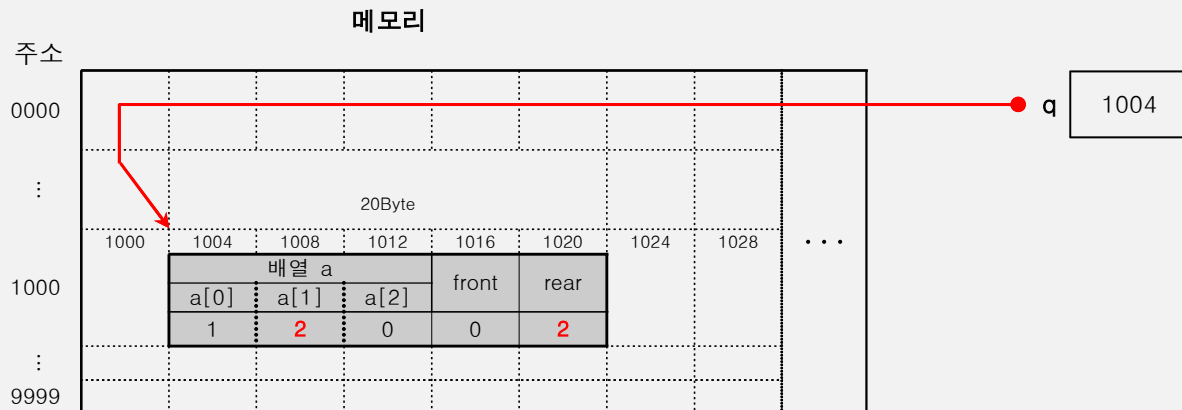
- ❸ `q`의 주소와 2를 인수로 `enq()` 함수를 호출한다.

```

7 void enq(Queue* q, int val) {
8   q -> a[q -> rear] = val;
9   q -> rear = (q -> rear + 1) % SIZE;
}

```

- ⑦ 반환값이 없는 enq() 함수의 시작점이다. ⑥번에서 전달한 배열 a의 주소를 q가, 2를 val이 받는다.
 ⑧ q가 가리키는 rear은 1이므로 q가 가리키는 a[1]에 val의 값 2를 저장한다.
 ⑨ q가 가리키는 rear에 '(q -> rear + 1) % SIZE', 즉 (1 + 1) % 3이므로 2를 저장한 후 제어를 enq() 함수를 호출했던 ⑥번의 다음 줄인 ⑩번으로 옮긴다.



```

int main( ) {
1  Queue q = {{0}, 0, 0};
2  enq(&q, 1);
6  enq(&q, 2);
10 deq(&q);
    enq(&q, 3);
    printf("%d 그리고 %d", deq(&q), deq(&q));
    return 0;
}

```

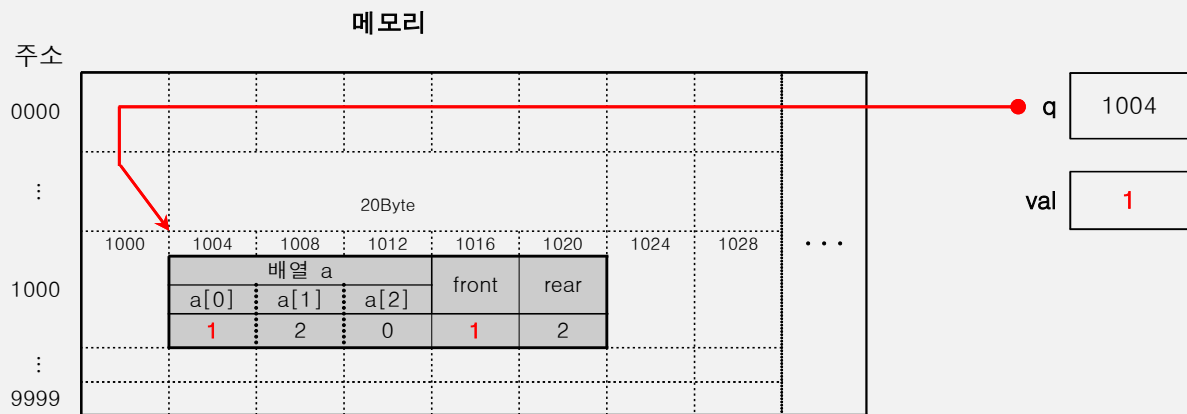
- ⑩ q의 주소를 인수로 deq() 함수를 호출한다.

```

11 int deq(Queue* q) {
12   int val = q -> a[q -> front];
13   q -> front = (q -> front + 1) % SIZE;
14   return val;
}

```

- ⑪ 정수를 반환하는 deq() 함수의 시작점이다. ⑩번에서 전달한 배열 a의 주소를 q가 받는다.
 ⑫ 정수형 변수 val을 선언하고, q가 가리키는 front가 0이므로 q가 가리키는 a[0]의 값 1로 초기화한다.
 ⑬ q가 가리키는 front에 '(q -> front + 1) % SIZE', 즉 (0 + 1) % 3이므로 1을 저장한다.
 ⑭ deq() 함수를 호출했던 곳으로 val을 반환하면서 제어를 ⑩번의 다음 줄인 ⑮번으로 옮긴다.



```
int main( ) {
  ❶ Queue q = {{0}, 0, 0};
  ❷ enq(&q, 1);
  ❸ enq(&q, 2);
  ❹ deq(&q);
  ❺ enq(&q, 3);
  printf("%d 그리고 %d", deq(&q), deq(&q));
  return 0;
}
```

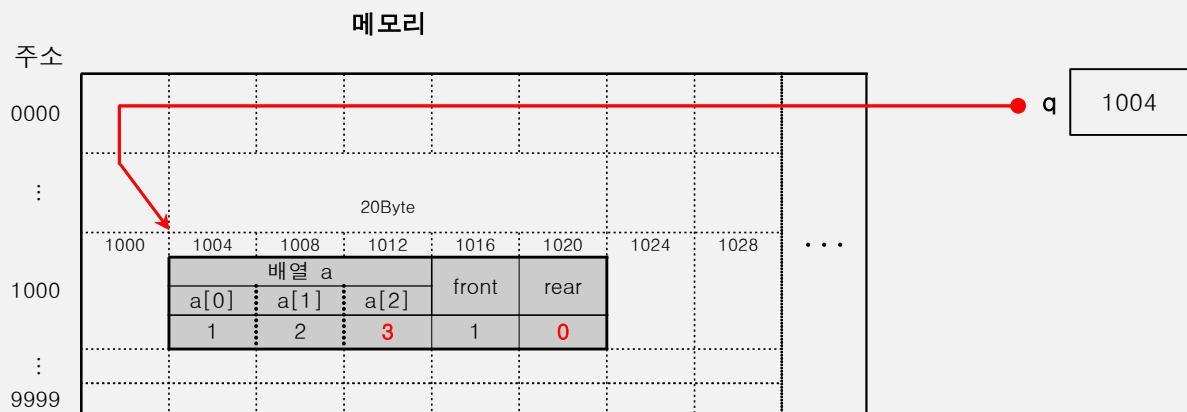
❺ q의 주소와 3을 인수로 enq() 함수를 호출한다.

```
❻ void enq(Queue* q, int val) {
  ❼ q -> a[q -> rear] = val;
  ❽ q -> rear = (q -> rear + 1) % SIZE;
}
```

❻ 반환값이 없는 enq() 함수의 시작점이다. ❺번에서 전달한 배열 a의 주소를 q가, 3을 val이 받는다.

❼ q가 가리키는 rear는 2이므로 q가 가리키는 a[2]에 val의 값 3을 저장한다.

❽ q가 가리키는 rear에 '(q -> rear + 1) % SIZE', 즉 (2 + 1) % 3이므로 0을 저장한 후 제어를 enq() 함수를 호출했던 ❺번의 다음 줄인 ❽번으로 옮긴다.



```

int main( ) {
❶ Queue q = {{0}, 0, 0};
❷ enq(&q, 1);
❸ enq(&q, 2);
❹ deq(&q);
❺ enq(&q, 3);
❻ printf("%d 그리고 %d", deq(&q), deq(&q));
   return 0;
}

```

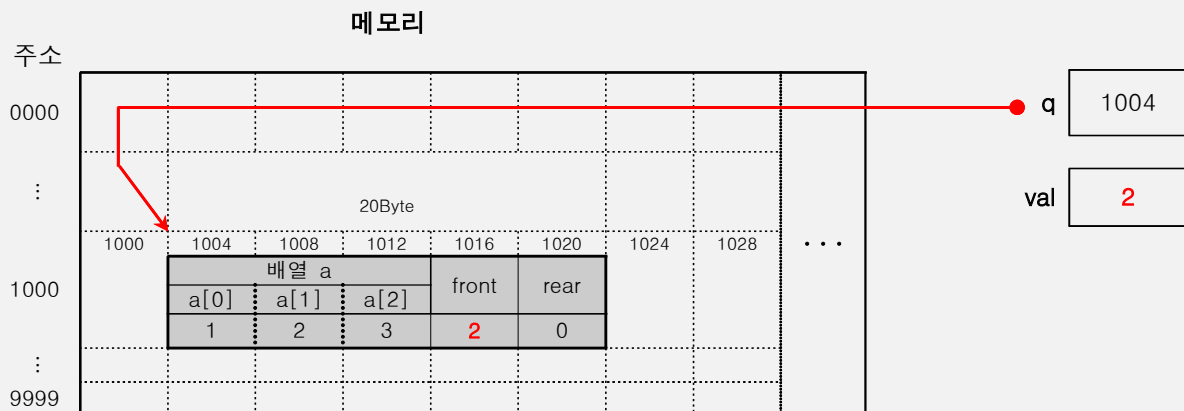
- ❹ q의 주소를 인수로 deq() 함수를 두 번 호출한 후 반환받은 값을 “%d 그리고 %d” 출력 서식에 적용하여 출력한다. 먼저 q의 주소를 인수로 deq() 함수를 호출한다.

```

❶ int deq(Queue* q) {
❷   int val = q -> a[q -> front];
❸   q -> front = (q -> front + 1) % SIZE;
❹   return val;
}

```

- ❶ 정수를 반환하는 deq() 함수의 시작점이다. ❹번에서 전달한 배열 a의 주소를 q가 받는다.
 ❷ 정수형 변수 val을 선언하고, q가 가리키는 front가 1이므로 q가 가리키는 a[1]의 값 2로 초기화한다.
 ❸ q가 가리키는 front에 '(q -> front + 1) % SIZE', 즉 (1 + 1) % 3이므로 2를 저장한다.
 ❹ deq() 함수를 호출했던 ❶번으로 val의 값 2를 반환한다.



```

int main( ) {
❶ Queue q = {{0}, 0, 0};
❷ enq(&q, 1);
❸ enq(&q, 2);
❹ deq(&q);
❺ enq(&q, 3);
❻❹ printf("%d 그리고 %d", deq(&q), deq(&q));
   return 0;
}

```

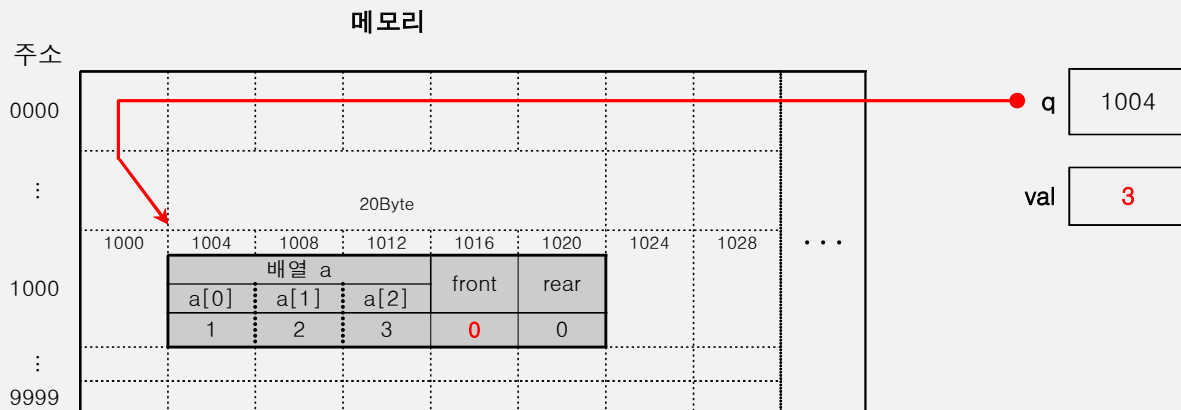
- ❹ 이어서 q의 주소를 인수로 deq() 함수를 다시 한 번 호출한다.

```

㉕ int deq(Queue* q) {
㉖   int val = q -> a[q -> front];
㉗   q -> front = (q -> front + 1) % SIZE;
㉘   return val;
}

```

- ㉕ 정수를 반환하는 deq() 함수의 시작점이다. ㉖번에서 전달한 배열 a의 주소를 q가 받는다.
 ㉖ 정수형 변수 val을 선언하고, q가 가리키는 front가 2이므로 q가 가리키는 a[2]의 값 3으로 초기화한다.
 ㉗ q가 가리키는 front에 '(q -> front + 1) % SIZE', 즉 (2 + 1) % 3이므로 0을 저장한다.
 ㉘ deq() 함수를 호출했던 ㉙번으로 val의 값 3을 반환한다.



```

int main( ) {
❶ Queue q = {{0}, 0, 0};
❷ enq(&q, 1);
❸ enq(&q, 2);
❹ deq(&q);
❺ enq(&q, 3);
❻❻❻ printf("%d 그리고 %d", deq(&q), deq(&q));
❼ return 0;
}

```

- ㉙ ㉙번으로부터 반환받은 2와 공백 한 칸을 출력하고 그리고를 출력한 후 공백 한 칸과 ㉙번으로부터 반환받은 3을 출력한다.

결과 2 그리고 3

- ㉙ main() 함수에서의 'return 0'은 프로그램의 종료를 의미한다.

[문제 11]

※ 다음 중 하나를 쓰면 됩니다.

AJAX, Asynchronous JavaScript and XML

※ 답안 작성 시 주의 사항 : 한글 또는 영문을 Full-name이나 약어로 쓰라는 지시사항이 없으면 한글이나 영문 약어로 쓰는 것이 유리합니다. 영문을 Full-name으로 풀어쓰다가 스펠링을 틀리면 오답으로 처리되니까요.

[문제 12]

Proxy

[문제 13]

SYN Flooding

[문제 14]

(1) → (2) → (3) → (4) → (5) → (6) → (7)

(1) → (2) → (4) → (5) → (6) → (1)

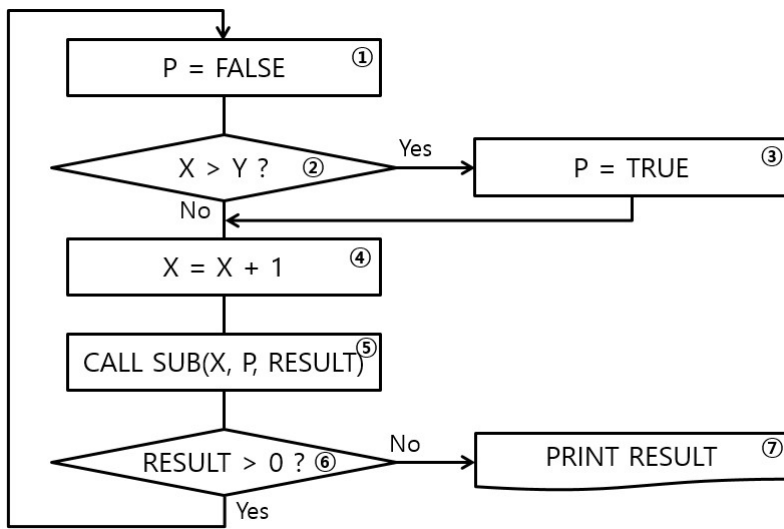
또는

(1) → (2) → (3) → (4) → (5) → (6) → (1)

(1) → (2) → (4) → (5) → (6) → (7)

[해설]

- 화이트박스 테스트의 검증 기준(Coverage) 중 분기 검증 기준(Branch Coverage)은 소스 코드의 모든 조건문이 한 번 이상 수행되도록 테스트 케이스를 설계하는 방법입니다.



- 위의 순서도를 기반으로 한 테스트 케이스는 ①번에서 시작한 프로세스가 조건문인 ②번과 ⑥번에 도달했을 때 반드시 한 번은 Yes로 한 번은 No로 진행되도록 설계되어야 합니다. 또한 문제지의 답란에 7칸의 괄호와 6칸의 괄호가 제시되어 있으므로, 두 번의 프로세스로 모든 코드가 수행되도록 설계해야 합니다.

[첫 번째 테스트 케이스 설계 방안]

- 7칸 괄호 : ①②③④⑤⑥⑦

- 6칸 괄호 : ①②④⑤⑥①

※ 7칸 괄호에 맞는 테스트 케이스를 설계할 때 ②번 조건문에서 Yes로, ⑥번 조건문에서 No로 진행되도록 설계했으므로, 6칸 괄호에 맞는 테스트 케이스는 ②번 조건문에서 No로, ⑥번 조건문에서 Yes로 진행되도록 설계해야 합니다.

[두 번째 테스트 케이스 설계 방안]

- 7칸 괄호 : ①②③④⑤⑥①

- 6칸 괄호 : ①②④⑤⑥⑦

※ 7칸 괄호에 맞는 테스트 케이스를 설계할 때 ②번 조건문에서 Yes로, ⑥번 조건문에서도 Yes로 진행되도록 설계했으므로, 6칸 괄호에 맞는 테스트 케이스는 ②번 조건문에서 No로, ⑥번 조건문에서도 No로 진행되도록 설계해야 합니다.

[문제 15]

11.75

[해설]

- RR(Round Robin)은 주어진 시간 할당량(Time Slice) 동안 실행되지 못할 경우 준비상태 큐의 가장 마지막으로 재배치하여 차례를 기다리므로 다음과 같이 표시할 수 있다.

진행 시간	0	4	8	12	16	20	24	25	26
프로세스	A	B	C	D	A	C	D	C	
실행 시간	4	4	4	4	4	4	1	1	

※ 색 동그라미는 프로세스가 완료됨을 표시한 것입니다(A → 20초, B → 8초, C → 26초, D → 25초).

• 대기 시간은 ‘종료 시간 - 도착 시간 - 실행 시간’으로 구할 수 있으므로 다음과 같다.

프로세스	종료 시간	도착 시간	실행 시간	대기 시간
A	20	0	8	12
B	8	1	4	3
C	26	2	9	15
D	25	3	5	17

∴ 평균 대기 시간은 $(12+3+15+17) / 4 = 11.75$

[문제 16]

5 그리고 6

[해설]

```
#include <stdio.h>

struct dat {           // 구조체 dat를 정의한다.
    int x;             // 정수형 변수 x를 선언한다.
    int y;             // 정수형 변수 y를 선언한다.
};

int main( ) {
    ❶ struct dat a[] = {{1, 2}, {3, 4}, {5, 6}};
    ❷ struct dat* ptr = a;
    ❸ struct dat** pptr = &ptr;
    ❹ (*pptr)[1] = (*pptr)[2];
    ❺ printf("%d 그리고 %d", a[1].x, a[1].y);
    ❻ return 0;
}
```

모든 C 프로그램은 반드시 main() 함수에서 시작한다.

❶ 3개의 요소를 갖는 dat 구조체 형태의 배열 a를 선언하고, 초기화한다. (이후 그림에서 지정한 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했음)



❷ dat 구조체 형태의 포인터 변수 ptr을 선언하고, a의 주소로 초기화한다. ptr은 a의 시작 주소를 가리킨다.

메모리

주소	0000						
	:						
	:	첫 번째		두 번째		세 번째	
	:	x	y	x	y	x	y
a	0500	1	2	3	4	5	6
	:	a[0]		a[1]		a[2]	
	:						
ptr	1000	500					
	:						
	:						
	2000						
	:						
	:						
	9999						

- ③ dat 구조체 형태의 이중 포인터 변수 pptr을 선언하고, ptr의 주소로 초기화한다. ptr이 a의 시작 주소를 가리키고 있으므로, 주소의 주소를 가리키는 pptr도 a의 시작 주소를 가리킨다.

메모리

주소	0000						
	:						
	:	첫 번째		두 번째		세 번째	
	:	x	y	x	y	x	y
a	0500	1	2	3	4	5	6
	:	a[0]		a[1]		a[2]	
	:						
ptr	1000	500					
	:						
pptr	2000	1000					
	:						
	:						
	9999						

- ④ (*pptr)[1]은 pptr이 가지고 있는 값 500이 가리키는 곳의 두 번째인 a[1]을 의미한다. 즉 a[1] = a[2]이므로 a 배열의 세 번째 값을 a 배열의 두 번째 값에 저장한다.

메모리

주소	0000						
	:						
	:	첫 번째		두 번째		세 번째	
	:	x	y	x	y	x	y
a	0500	1	2	5	6	5	6
	:	a[0]		a[1]		a[2]	
	:						
ptr	1000	500					
	:						
pptr	2000	1000					
	:						
	:						
	9999						

- ⑤ a[1]의 x인 5와 공백 한 칸을 출력하고 그리고를 출력한 후 공백 한 칸과 a[1]의 y인 6을 출력한다.

결과 5 그리고 6

- ⑥ main() 함수에서의 'return 0'은 프로그램의 종료를 의미한다.

[문제 17]

2

[해설]

```

❶ lst = [1,2,3]
❷ dst = {i : i * 2 for i in lst}
❸ s = set(dst.values( ))
❹ lst[0] = 99
❺ dst[2] = 7
❻ s.add(99)
❼ print(len(s & set(dst.values( ))))

```

- ❶ 리스트 lst를 선언하면서 초기값을 지정한다. 초기값으로 지정된 수만큼 리스트의 요소가 만들어진다.

	[0]	[1]	[2]
리스트 lst	1	2	3

- ❷ 딕셔너리 dst를 선언하고, 컴프리헨션을 적용한 for문에 의해 반환되는 i는 키로, i*2는 값으로 초기화한다.

	'1'	'2'	'3'
딕셔너리 dst	2	4	6

- for i in lst : 리스트 lst의 각 요소를 반복 변수 i에 저장하면서 리스트 lst의 요소 수만큼 반복한다.

※ **컴프리헨션(Comprehension)** : for 문의 반복 변수를 for 문 앞에 기술하여, 반복 변수의 값을 리스트나 딕셔너리의 형태로 반환함

- ❸ 세트 s를 선언하고, 딕셔너리 dst에서 값만 추출한 후 세트로 변환한 값으로 초기화한다.

	[0]	[1]	[2]
세트 s	2	4	6

- values() : 딕셔너리의 모든 값을 반환함
- set() : 리스트, 딕셔너리, 문자열 등을 세트로 변환함

- ❹ lst[0]에 99를 저장한다.

	[0]	[1]	[2]
리스트 lst	99	2	3

- ❺ 딕셔너리 dst에서 키가 2인 요소의 값을 7로 변경한다.

	'1'	'2'	'3'
딕셔너리 dst	2	7	6

- ❻ 세트 s에 99를 추가한다.

	[0]	[1]	[2]	[3]
세트 s	2	4	6	99

- add() : 세트에 값을 추가함

- ❼ 'len(s & set(dst.values()))'의 결과인 2를 출력한다.

len(s & set(dst.values()))

_____ ㉠
 _____ ㉡
 _____ ㉢

- ㉠ : 딕셔너리 dst에서 값만 추출한 후 세트로 변환한 값을 반환함 → {2, 7, 6}
- ㉡ : 세트 s와 세트 ㉠의 공통 요소(교집합(&))를 반환함 → {2, 4, 6, 99} & {2, 7, 6} → {2, 6}
- ㉢ : ㉡의 길이인 2를 반환함

결과 2

[문제 18]

5P

[해설]

```
public class Main {  
    public static class Parent {  
        public int x(int i) { return i + 2; }  
        ⑤      public static String id( ) { return "P"; }  
    }  
    public static class Child extends Parent {  
        ③      public int x(int i) { return i + 3; }  
        public String x(String s) { return s + "R"; }  
        public static String id( ) { return "C"; }  
    }  
    public static void main(String[] args) {  
        ①      Parent ref = new Child( );  
        ②④⑥    System.out.println(ref.x(2) + ref.id( ));  
    }  
}
```

모든 Java 프로그램은 반드시 main() 메소드에서 시작한다.

- ① Child 클래스의 생성자를 이용하여 Parent 클래스의 객체 변수 ref를 선언한다.
 - [부모클래스명] [객체변수명] = new [자식클래스생성자()] : 부모 클래스의 객체 변수를 선언하면서 자식 클래스의 생성자를 사용하면 형 변환이 발생한다.
 - 이렇게 형 변환이 발생했을 때 부모 클래스와 자식 클래스에 동일한 속성이나 메소드가 있으면 자식 클래스의 속성이나 메소드로 재정의된다.
- ② ref.x(2) 메소드를 호출한 후 반환받은 값과 ref.id() 메소드를 호출한 후 반환받은 값의 합을 출력한다. 먼저 ref의 x(2) 메소드를 호출한다. 2를 인수로 하여 ③번으로 이동한다.
 - ※ ①번에서 형 변환이 발생했으므로 부모 클래스와 자식 클래스에 동일한 속성이나 메소드가 있으면 자식 클래스의 속성이나 메소드로 재정의되는데, 인수가 정수이므로 자식 클래스의 x(int i) 메소드가 호출됩니다.
- ③ 정수를 반환하는 x() 메소드의 시작점이다. ②번에서 전달한 2를 i가 받아 'i+3'의 결과인 5를 메소드를 호출했던 ④번으로 반환한다.
- ④ 이어서 ref.id() 메소드를 호출한다. ⑤번으로 이동한다.
 - ※ id() 메소드는 static으로 선언된 정적 메소드입니다. 정적 메소드는 컴파일 시점에 객체 변수가 부모 클래스의 타입으로 결정되므로, Parent의 id() 메소드가 호출됩니다.
- ⑤ 문자를 반환하는 id() 메소드의 시작점이다. "P"를 메소드를 호출했던 ⑥번으로 반환한다.
- ⑥ ③번으로부터 반환받은 5에 ⑤번으로부터 반환받은 "P"를 덧붙여 출력한 후 커서를 다음 줄 처음으로 이동한다.

결과 5P

[문제 19]

1a3b3

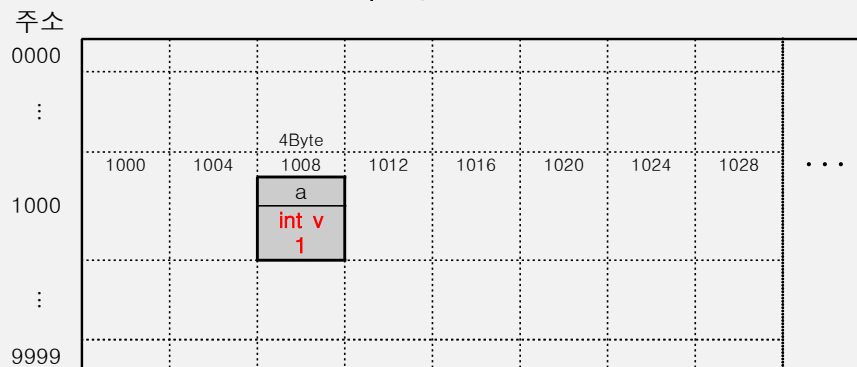
[해설]

```
public class Main {
    public static class BO {
        public int v;
        ②⑤⑧ public BO(int v) {
        ③⑥⑨         this.v = v;
        }
    }
    public static void main(String[] args) {
        ①      BO a = new BO(1);
        ④      BO b = new BO(2);
        ⑦      BO c = new BO(3);
        ⑩      BO[] arr = {a, b, c};
        ⑪      BO t = arr[0];
        ⑫      arr[0] = arr[2];
        ⑬      arr[2] = t;
        ⑭      arr[1].v = arr[0].v;
        ⑮      System.out.println(a.v + "a" + b.v + "b" + c.v);
    }
}
```

모든 Java 프로그램은 반드시 main() 메소드에서 시작한다.

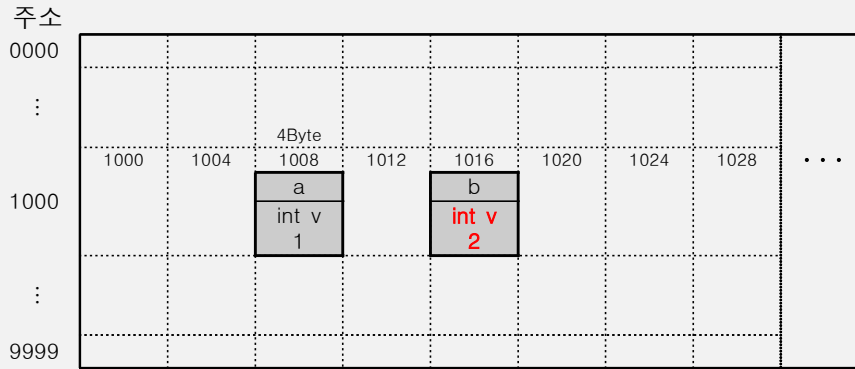
- ① 클래스 BO의 객체 변수 a를 선언하고, 생성자에 인수 1을 전달한다. ②번으로 이동한다.
- ② BO 클래스의 생성자 BO()의 시작점이다. ①번에서 전달받은 1을 정수형 변수 v가 받는다.
- ③ 현재 클래스(BO 클래스)의 v에 1을 저장한다. 생성자가 종료되면 호출했던 ①번의 다음 줄인 ④번으로 제어를 옮긴다. (이후 그림에서 객체 변수 a가 저장된 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했음)

메모리



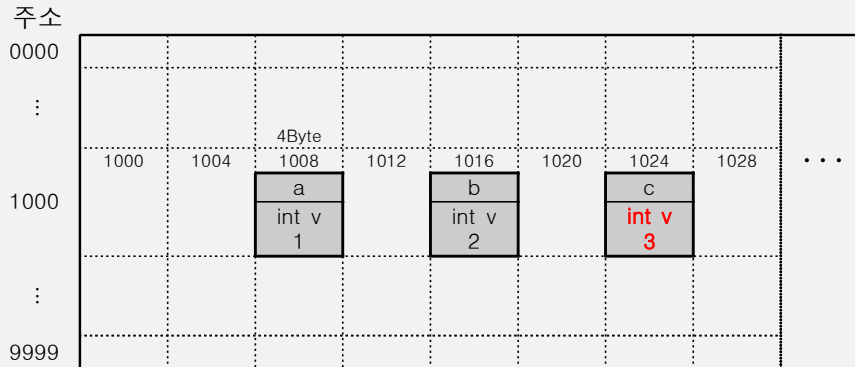
- ④ 클래스 BO의 객체 변수 b를 선언하고, 생성자에 인수 2를 전달한다. ⑤번으로 이동한다.
- ⑤ BO 클래스의 생성자 BO()의 시작점이다. ④번에서 전달받은 2를 정수형 변수 v가 받는다.
- ⑥ 현재 클래스(BO 클래스)의 v에 2를 저장한다. 생성자가 종료되면 호출했던 ④번의 다음 줄인 ⑦번으로 제어를 옮긴다.

메모리



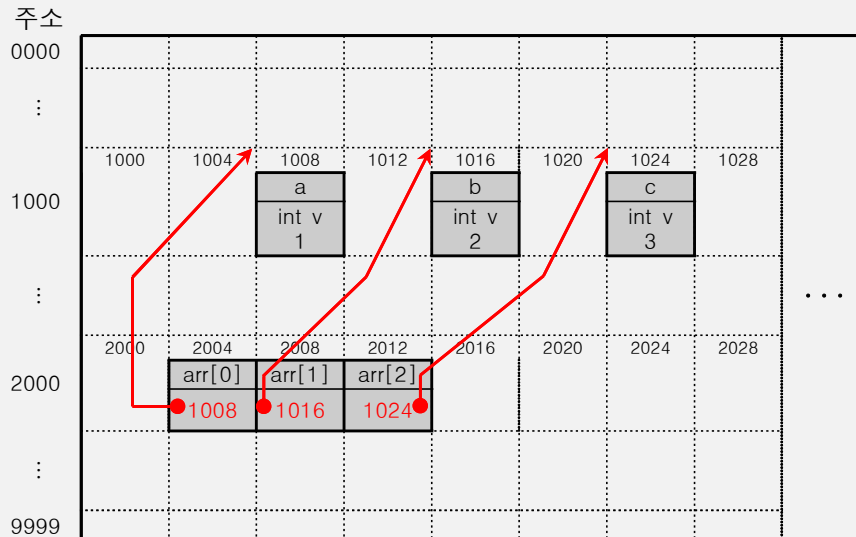
- ⑦ 클래스 BO의 객체 변수 `c`를 선언하고, 생성자에 인수 3을 전달한다. ⑧번으로 이동한다.
- ⑧ BO 클래스의 생성자 BO()의 시작점이다. ⑦번에서 전달받은 3을 정수형 변수 `v`가 받는다.
- ⑨ 현재 클래스(BO 클래스)의 `v`에 3을 저장한다. 생성자가 종료되면 호출했던 ⑦번의 다음 줄인 ⑩번으로 제어를 옮긴다.

메모리

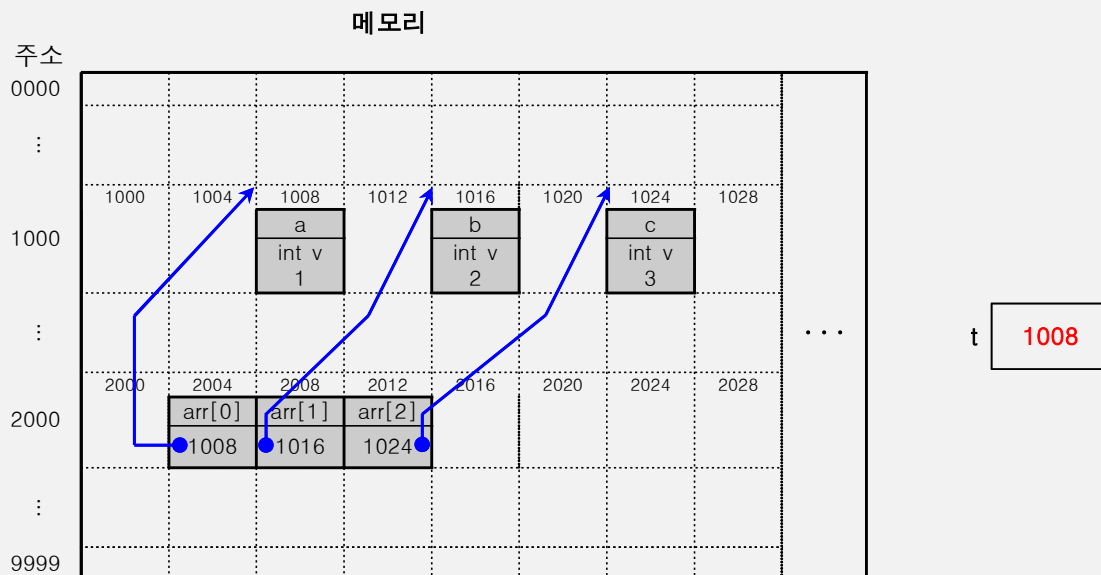


- ⑩ 3개의 요소를 갖는 BO 클래스 배열 `arr`를 선언하고 객체 변수 `a`, `b`, `c`로 초기화한다. `arr[0]`은 객체 변수 `a`의 주소를, `arr[1]`은 객체 변수 `b`의 주소를, `arr[2]`는 객체 변수 `c`의 주소를 가리키게 된다.

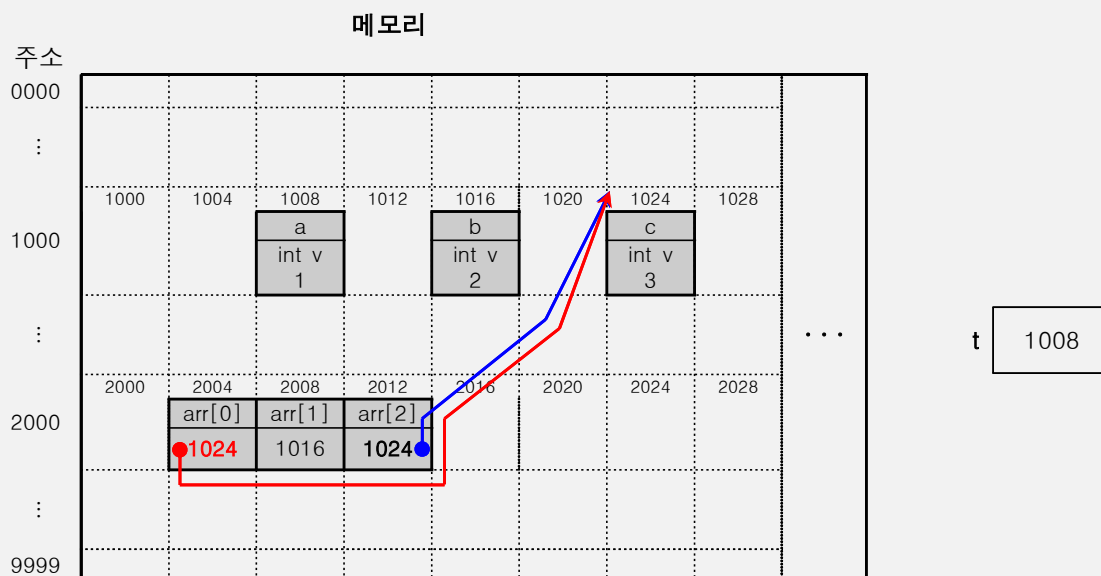
메모리



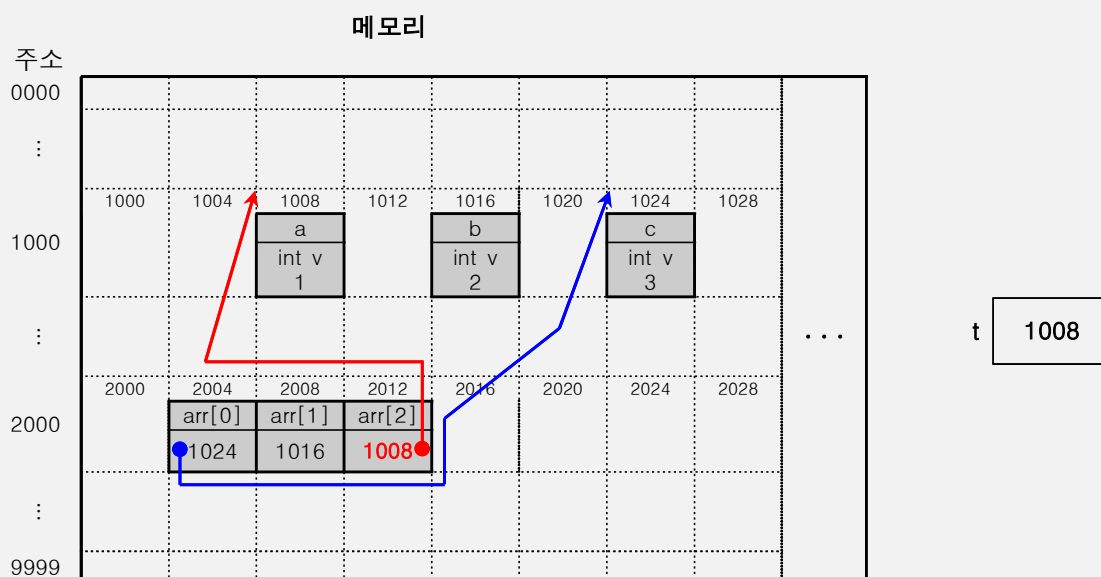
- ⑪ 클래스 BO 자료형 변수 `t`를 선언하고 `arr[0]`을 저장한다.



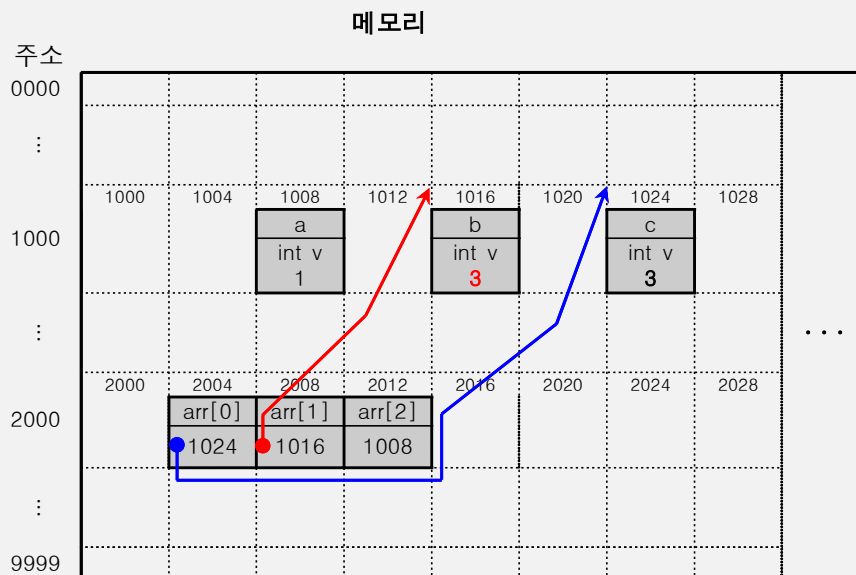
⑫ arr[0]에 arr[2]를 저장한다. 이제 arr[0]과 arr[2] 모두 객체 변수 c의 주소를 가리키게 된다.



⑬ t를 arr[2]에 저장한다.



⑭ arr[0]의 v를 arr[1]의 v에 저장한다.



⑮ a.v의 값, “a” 문자, b.v의 값, “b” 문자, c.v의 값을 모두 붙여 출력한다.

결과 1a3b3

[문제 20]

TSEB

[해설]

```
#include <stdio.h>
#include <stdlib.h>

struct node {
    char c;
    struct node* p;
};
```

// 구조체 node를 정의한다.
// 문자형 변수 c를 선언한다.
// node의 구조체 포인터 변수 p를 선언한다.

struct node	
char c (1Byte)	struct node* p (4Byte)
데이터를 저장할 멤버	다음 노드의 주소를 저장할 멤버

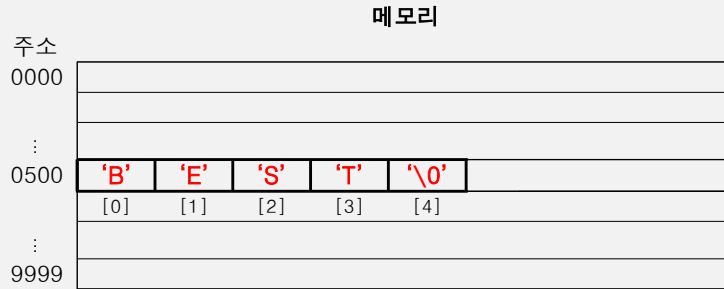
```
int main( ) {
    ① struct node* n = func("BEST");
    while(n) {
        putchar(n -> c);
        struct node* t = n;
        n = n -> p;
        free(t);
    }
    return 0;
}
```

모든 C 언어 프로그램은 반드시 main() 함수에서 시작한다.

① node 자료형 포인터 변수 n을 선언하고, “BEST”를 인수로 func() 함수를 호출한 후 반환받은 값으로 초기화한다.

※ 인수로 문자열을 지정하면 메모리의 저장된 문자열의 시작 주소가 전달됩니다. 이후 그림에서 지정한

주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했습니다.



```

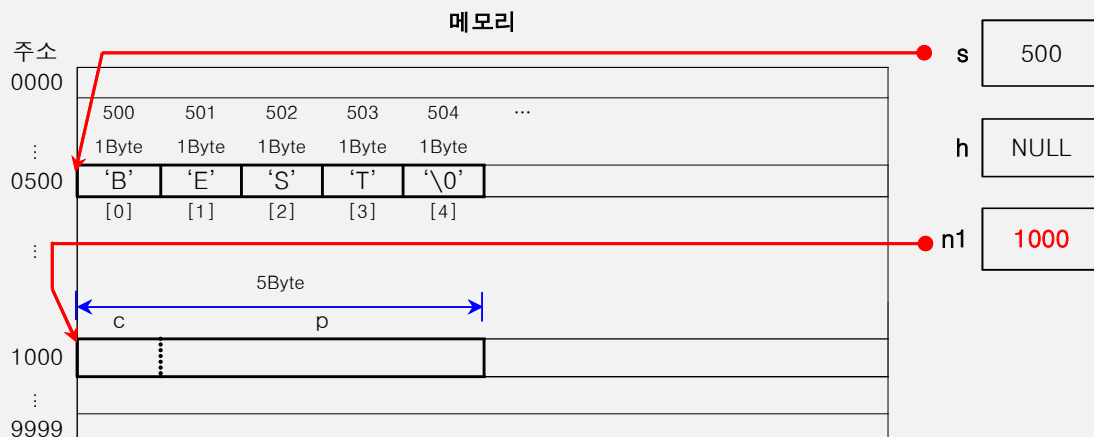
② struct node* func(char* s) {
③     struct node* h = NULL, *n;
④     while(*s) {
⑤         n = malloc(sizeof(struct node));
⑥         n -> c = *s++;
⑦         n -> p = h;
⑧         h = n;
    }
    return h;
}

```

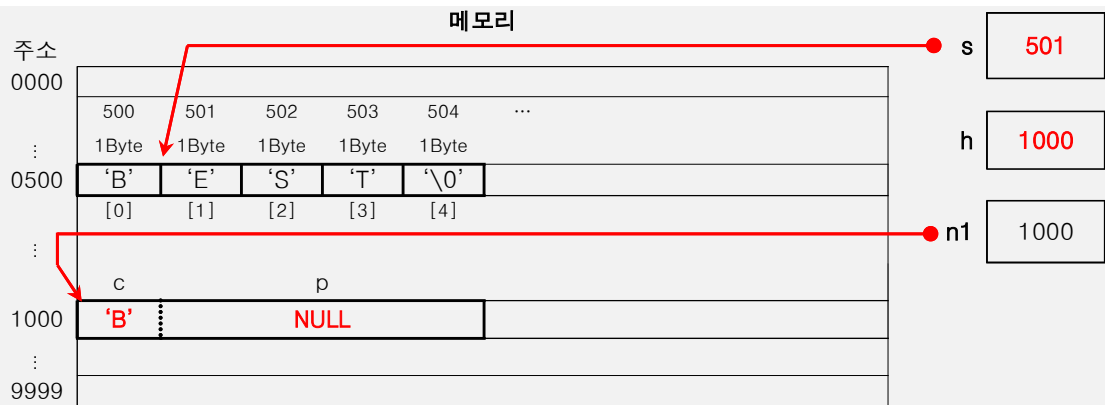
- ② node 자료형 포인터를 반환하는 func() 함수의 시작점이다. ①번에서 전달한 “BEST” 문자열의 시작 주소를 문자형 포인터 변수 s가 받는다.
- ③ node 자료형 포인터 변수 h와 n을 선언하고, h는 NULL로 초기화한다.
- ④ s가 가리키는 값이 NULL이 아닌 동안 ⑤~⑧번을 반복 수행한다.

[첫 번째 반복]

- ⑤ malloc 함수가 메모리에서 node 자료형의 크기, 즉 5Byte의 빈 영역을 찾아 할당한 다음 그 영역의 시작 주소를 포인터 변수 n에 저장한다. 이제 n은 할당된 공간의 시작 주소를 가리킨다.
 - ※ malloc 함수가 동적으로 할당하는 것이므로 여기서 지정한 주소는 임의로 정한 것이며, 이해를 돕기 위해 주소를 10진수로 표현했습니다. 또한 malloc 함수는 메모리 크기를 바이트 단위로 지정합니다.
 - ※ 동적 주소가 저장되는 n은 반복되는 동안 매번 새롭게 만들어지는 것으로, 구분하기 쉽게 n1, n2, n3, ...과 같이 일련번호를 부여하겠습니다.

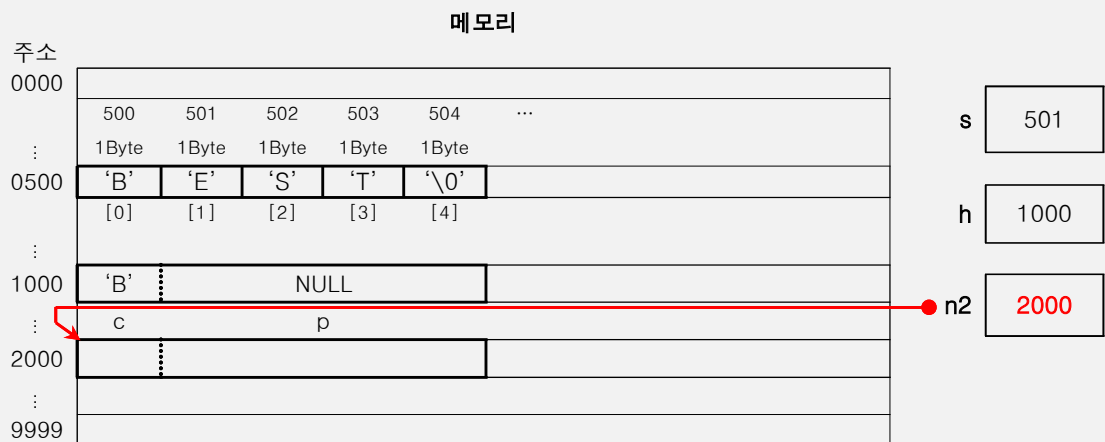


- ⑥ s가 가리키는 곳의 값 'B'를 n이 가리키는 곳의 c에 저장한 후 s의 값을 1 증가시킨다.
- ⑦ h의 값 NULL을 n이 가리키는 곳의 p에 저장한다.
- ⑧ n의 값을 h에 저장한다.

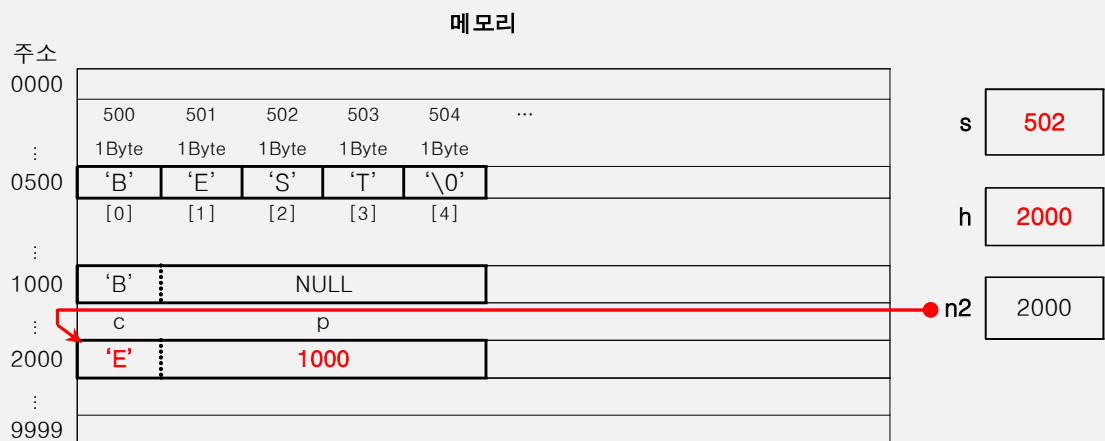


[두 번째 반복]

- ⑤ malloc 함수가 메모리에서 node 자료형의 크기, 즉 5Byte의 빈 영역을 찾아 할당한 다음 그 영역의 시작 주소를 포인터 변수 n에 저장한다. 이제 n은 할당된 공간의 시작 주소를 가리킨다.

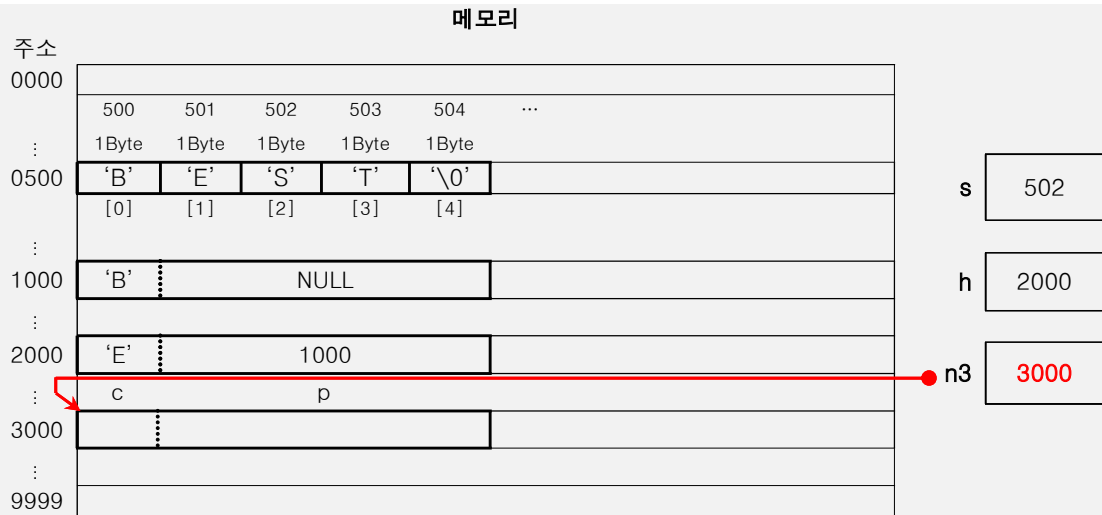


- ⑥ s가 가리키는 곳의 값 'E'를 n이 가리키는 곳의 c에 저장한 후 s의 값을 1 증가시킨다.
 ⑦ h의 값 1000을 n이 가리키는 곳의 p에 저장한다.
 ⑧ n의 값을 h에 저장한다.

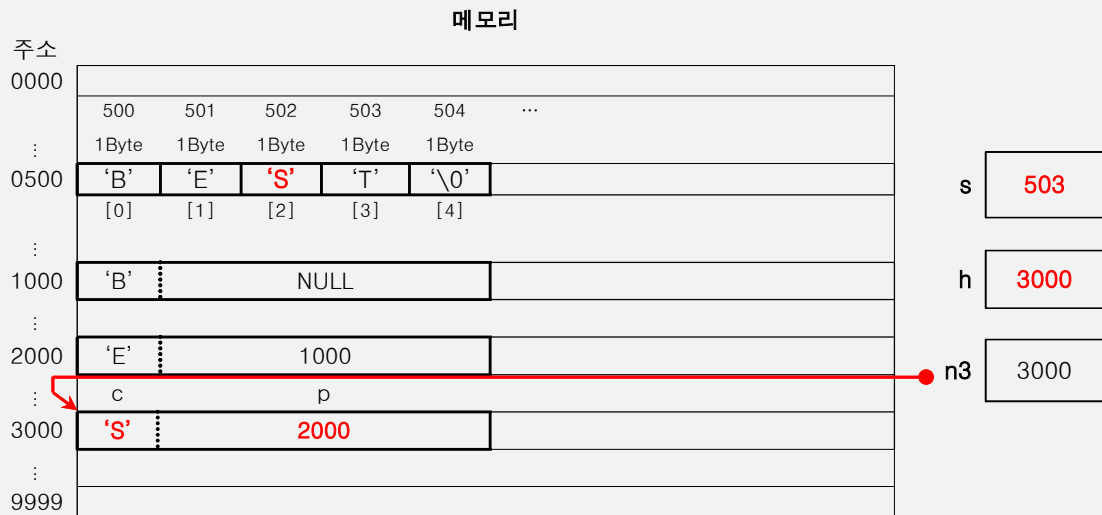


[세 번째 반복]

- ⑤ malloc 함수가 메모리에서 node 자료형의 크기, 즉 5Byte의 빈 영역을 찾아 할당한 다음 그 영역의 시작 주소를 포인터 변수 n에 저장한다. 이제 n은 할당된 공간의 시작 주소를 가리킨다.

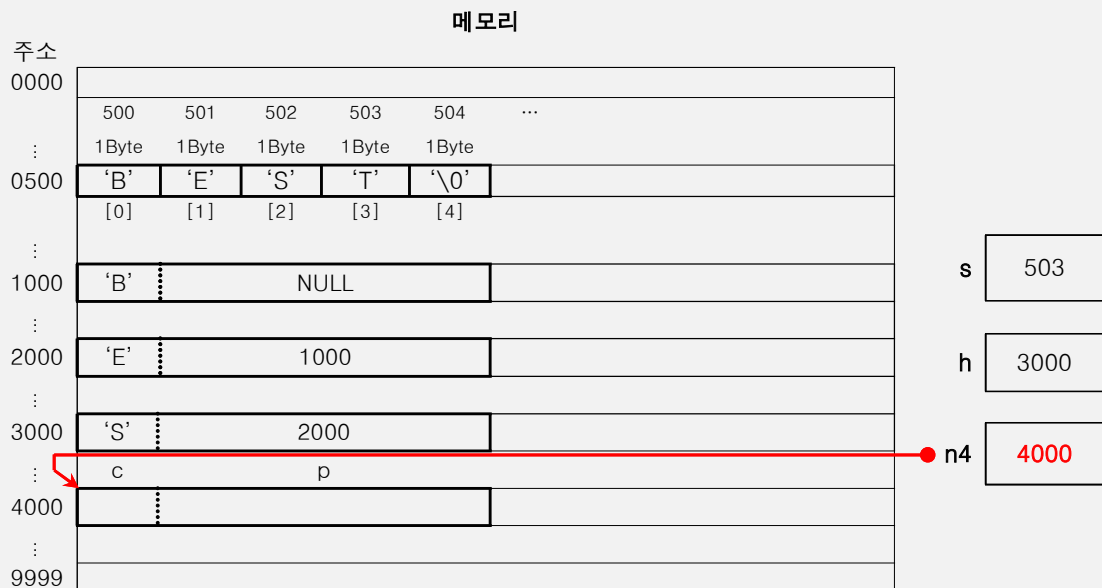


- ⑥ s가 가리키는 곳의 값 'S'를 n이 가리키는 곳의 c에 저장한 후 s의 값을 1 증가시킨다.
- ⑦ h의 값 2000을 n이 가리키는 곳의 p에 저장한다.
- ⑧ n의 값을 h에 저장한다.



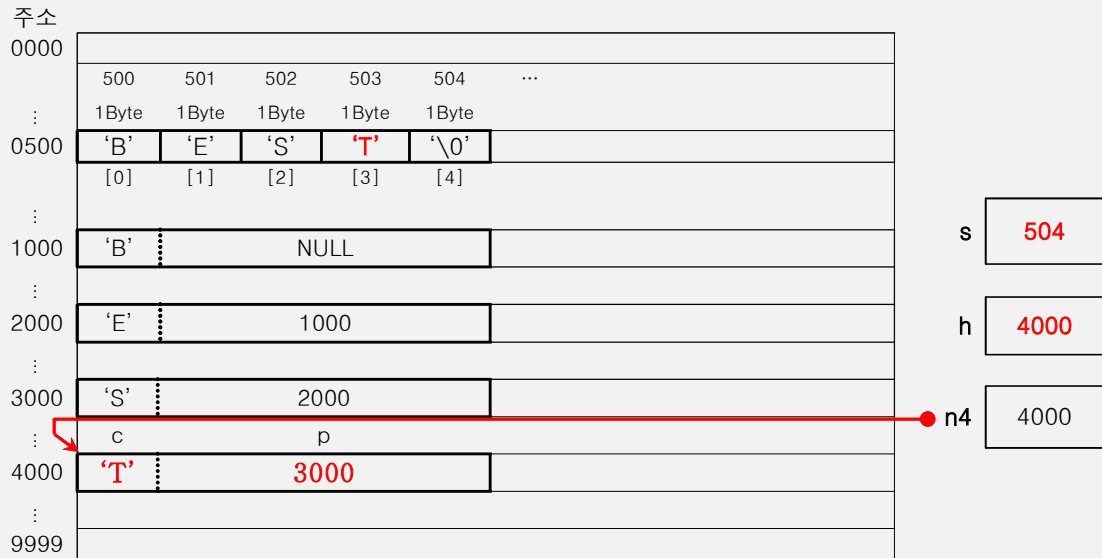
[네 번째 반복]

- ⑤ malloc 함수가 메모리에서 node 자료형의 크기, 즉 5Byte의 빈 영역을 찾아 할당된 다음 그 영역의 시작 주소를 포인터 변수 n에 저장한다. 이제 n은 할당된 공간의 시작 주소를 가리킨다.



- ⑥ s가 가리키는 곳의 값 'T'를 n이 가리키는 곳의 c에 저장한 후 s의 값을 1 증가시킨다.
- ⑦ h의 값 3000을 n이 가리키는 곳의 p에 저장한다.
- ⑧ n의 값을 h에 저장한다.

메모리



```

② struct node* func(char* s) {
③     struct node* h = NULL, *n;
④     while(*s) {
⑤         n = malloc(sizeof(struct node));
⑥         n -> c = *s++;
⑦         n -> p = h;
⑧         h = n;
⑨     }
⑩     return h;
}

```

- ④ s가 가리키는 값이 '\0', 즉 NULL이므로 반복을 중단하고 ⑨번으로 이동한다.
- ⑨ h가 가리키는 주소를 func() 함수를 호출했던 ⑩번으로 반환한다.

```

int main( ) {
①⑩ struct node* n = func("BEST");
⑪ while(n) {
⑫     putchar(n -> c);
⑬     struct node* t = n;
⑭     n = n -> p;
⑮     free(t);
}
return 0;
}

```

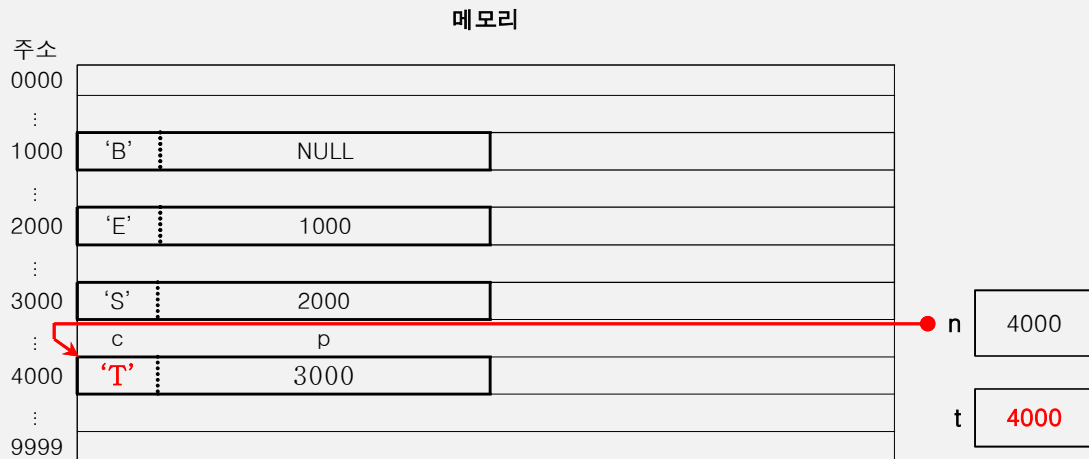
- ⑩ ⑨번으로부터 반환받은 h가 가리키는 주소 4000을 node 자료형 포인터 변수 n에 저장한다.
- ⑪ n이 NULL이 아닌 동안 ⑫~⑮번을 반복 수행한다.

[첫 번째 반복]

- ⑫ n이 가리키는 c의 값 'T'를 출력한다.

결과 T

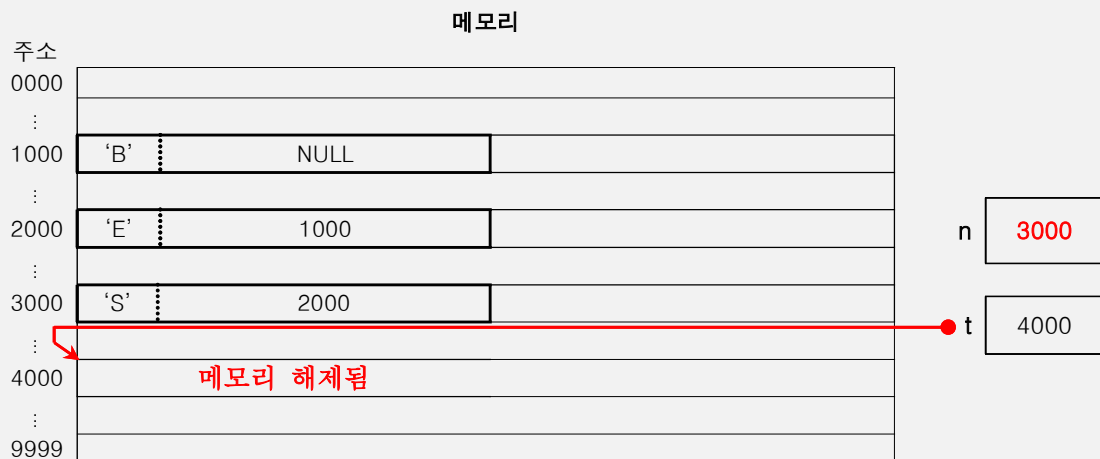
- ⑬ node 자료형 포인터 변수 t를 선언하고, n으로 초기화한다.



- ⑭ n이 가리키는 곳의 p의 값을 n에 저장한다.

- ⑮ t가 가리키고 있는 메모리 공간을 해제한다. 즉 t가 가리키고 있는 4000 번지부터 할당된 5Byte 공간을 해제한다.

※ free() : malloc() 함수에 의해 동적으로 할당된 메모리를 해제함

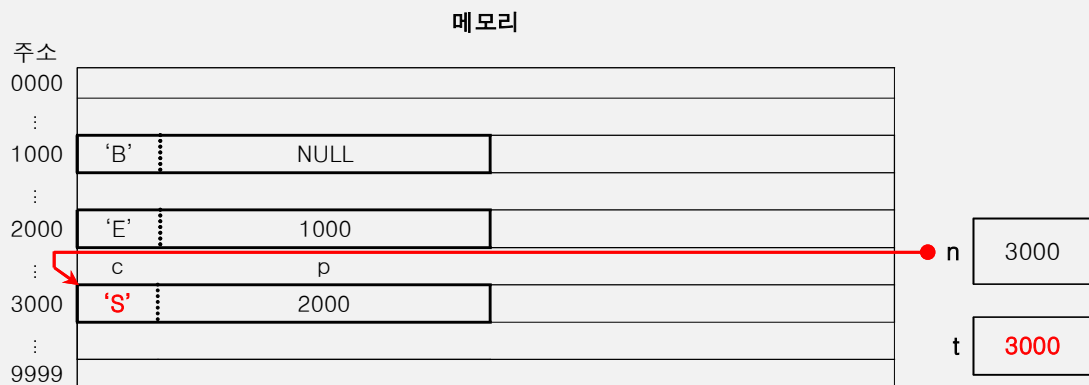


[두 번째 반복]

- ⑫ n이 가리키는 곳의 c의 값 'S'를 출력한다.

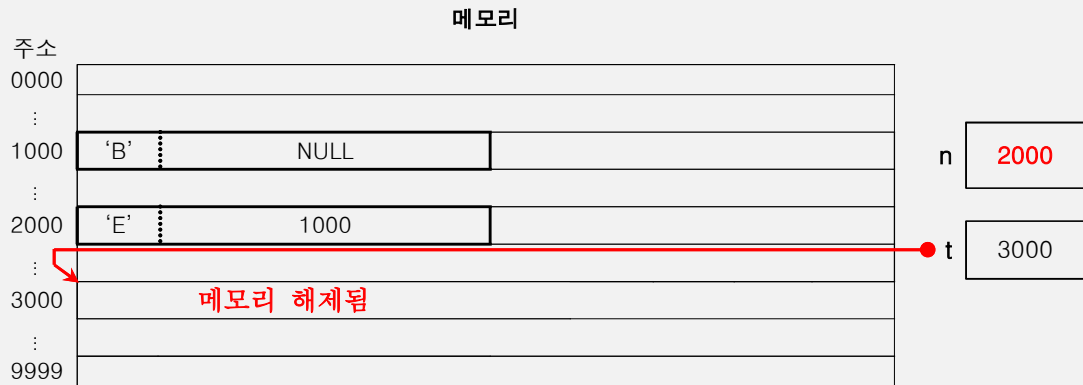
결과 TS

- ⑬ node 자료형 포인터 변수 t를 선언하고, n으로 초기화한다.



- ⑭ n이 가리키는 곳의 p의 값을 n에 저장한다.

- ⑮ t가 가리키고 있는 메모리 공간을 해제한다. 즉 t가 가리키고 있는 3000 번지부터 할당된 5Byte 공간을 해제한다.

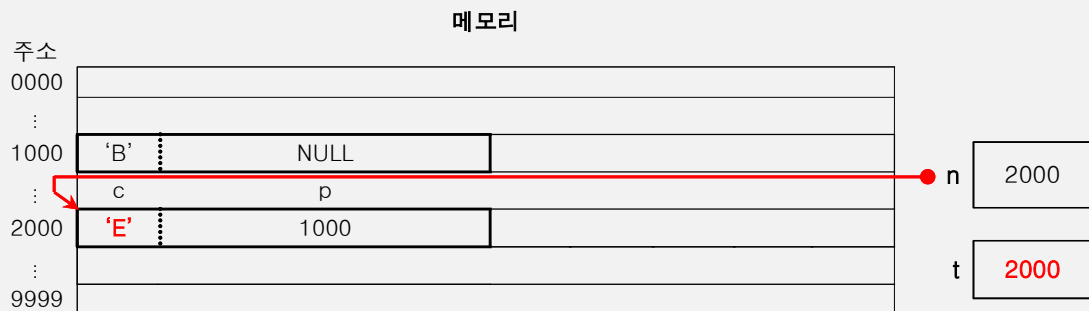


[세 번째 반복]

- ⑫ n이 가리키는 c의 값 'E'를 출력한다.

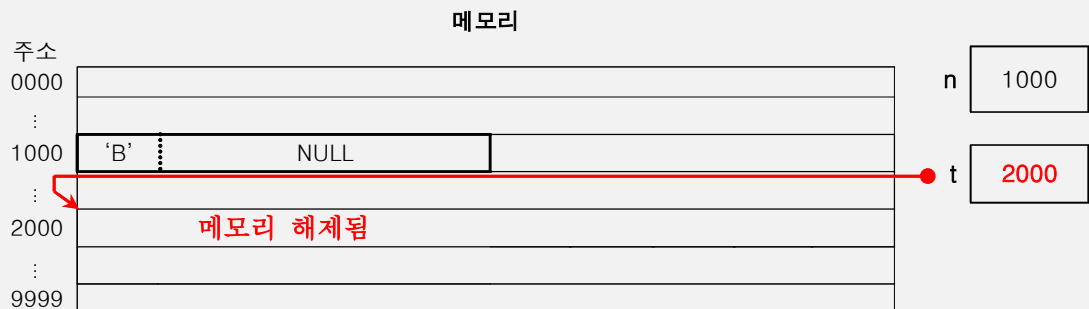
결과 TSE

- ⑬ node 자료형 포인터 변수 t를 선언하고, n으로 초기화한다.



- ⑭ n이 가리키는 곳의 p의 값을 n에 저장한다.

- ⑮ t가 가리키고 있는 메모리 공간을 해제한다. 즉 t가 가리키고 있는 2000 번지부터 할당된 5Byte 공간을 해제한다.

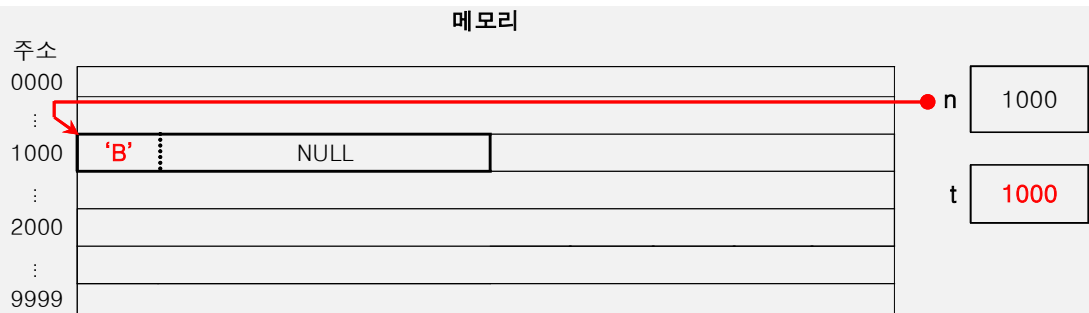


[네 번째 반복]

- ⑫ n이 가리키는 c의 값 'B'를 출력한다.

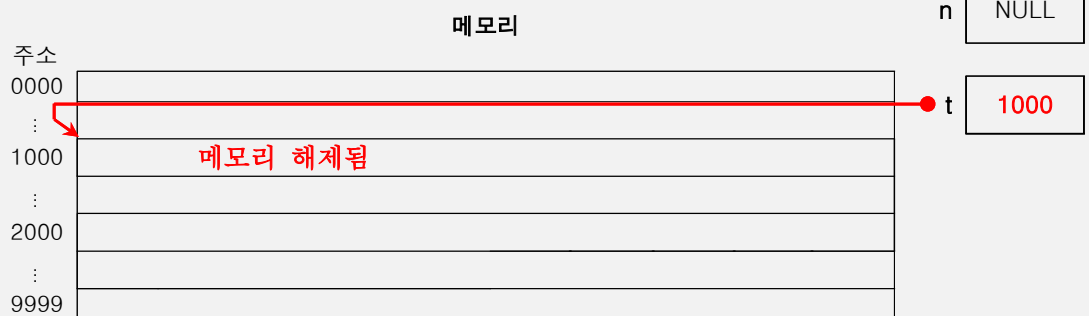
결과 TSEB

- ⑬ node 자료형 포인터 변수 t를 선언하고, n으로 초기화한다.



⑭ n이 가리키는 곳의 p의 값을 n에 저장한다.

⑮ t가 가리키고 있는 메모리 공간을 해제한다. 즉 t가 가리키고 있는 1000 번지부터 할당된 5Byte 공간을 해제한다.



```

int main( ) {
  ①⑩ struct node* n = func("BEST");
  ⑪ while(n) {
    ⑫     putchar(n -> c);
    ⑬     struct node* t = n;
    ⑭     n = n -> p;
    ⑮     free(t);
  }
  ⑯ return 0;
}

```

⑪ n이 NULL이므로 반복을 중단하고 ⑯번으로 이동한다.

⑯ main() 함수에서의 'return 0'은 프로그램의 종료를 의미한다.