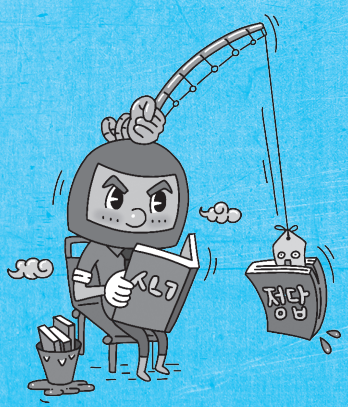
부록

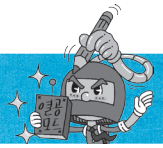
1과목 실무 알고리즘 응용
정답 및 해설

1과목 정답 및 해설

실무 알고리즘 응용



1장 정답 소프트웨어 개발의 기초



Section 001

[문제 1]

객체지향 기법

[문제 2]

클래스(Class)

[문제 3]

객체들 간에 상호작용을 하는 데 사용되는 수단으로, 객체에게 어떤 행위를 하도록 지시하는 명령 또는 요구 사항이다.

[문제 4]

- **데이터** : 객체가 가지고 있는 정보로 속성이나 상태, 분류 등을 나타낸다.
- **메소드** : 객체가 수행하는 기능으로, 객체가 갖는 데이터를 처리하는 알고리즘이다.

Section 002

[문제 1]

캡슐화(Encapsulation)

[문제 2]

다른 객체에게 자신의 정보를 숨기고 자신의 연산만을 통하여 접근을 허용하는 것이다.

[문제 3]

추상화(Abstraction)

[문제 4]

다형성(Polymorphism)

[문제 5]

이미 정의된 상위 클래스의 모든 속성과 연산을 하위 클래스가 물려받는 것이다.

Section 003

[문제 1]

객체지향 구현

[문제 2]

① 계획 및 분석 ② 설계 ③ 구현 ④ 테스트 및 검증

[문제 3]

- **클래스 테스트** : 구조적 기법에서의 단위 테스트와 같은 개념으로, 가장 작은 단위, 즉 캡슐화된 클래스나 객체를 검사하는 것이다.
- **통합 테스트** : 객체를 몇 개 결합하여 하나의 시스템으로 완성시키는 과정에서의 검사이다.
- **확인 테스트** : 사용자 요구사항에 대한 만족 여부를 검사한다.
- **시스템 테스트** : 모든 요소들이 적합하게 통합되고 올바른 기능을 수행하는지 검사한다.

[문제 4]

객체지향 분석

[문제 5]

문제 정의, 요구 명세화, 객체 연산자 정의, 객체 인터페이스 결정, 객체 구현

Section 004

[문제 1]

- **개념** : 상호작용 애플리케이션을 모델(Model), 뷰(View), 컨트롤러(Controller)의 세 개의 컴포넌트로 구분하는 아키텍처로, 유저 인터페이스와 비즈니스 로직들을 서로 분리하여 개발하는 방법이다.
- **장점** : 동일한 모델에 대해 다양한 뷰를 제공하고, 모델과 뷰의 구분으로 사용자 인터페이스에 대한 요구 사항을 적용시키는데 용이하다.

[문제 2]

① 표준화 ② 중립성 ③ 유연성 ④ 의사소통

[문제 3]

계층 구조(Layered Architecture)

[문제 4]

새로운 서버의 추가 및 업그레이드가 용이하다. 데이터가 서버에 집중되어 있어 데이터 관리가 용이하며, 보안적인 측면이 뛰어나다. 서버에 네트워크 트래픽과 데이터가 집중되어 처리 비용이 급증할 수 있다.

[문제 5]

데이터의 흐름을 점진적으로 처리하는 시스템을 위한 구조를 제공하는 아키텍처이다. 프로세싱을 위한 시스템이 각 필터에 캡슐화 되어 있으며, 데이터는 인접 필터 사이의 파이프를 통해 전달되는 형태이다.

[문제 6]

- M : Model
- V : View
- C : Controller

예상문제은행

1장 소프트웨어 개발의 기초

[문제 1]

객체지향 설계

[문제 2]

현실 세계의 개체를 기계의 부품처럼 하나의 객체로 만들어, 기계적인 부품들을 조립하여 제품을 만들 듯이 소프트웨어를 개발할 때에도 객체들을 조립해서 작성할 수 있도록 하는 기법이다.

[문제 3]

① – ⓓ ② – ⓒ ③ – ⓐ ④ – ⓔ ⑤ – ⓑ

[문제 4]

객체지향 프로그래밍

[문제 5]

- **객체(Object)** : 데이터와 데이터를 처리하는 함수를 묶어 놓은 하나의 소프트웨어 모듈이다.
- **클래스(Class)** : 공통된 속성과 연산을 갖는 객체의 집합으로, 객체의 일반적인 타입을 의미한다.
- **메시지(Message)** : 객체들 간에 상호작용을 하는 데 사용되는 수단으로, 객체에게 어떤 행위를 하도록 지시하는 명령 또는 요구 사항이다.

[문제 6]

① 클래스 기반 언어 ② 객체 지향성 언어 ③ 객체 기반 언어

[문제 7]

① – ⓓ ② – ⓑ ③ – ⓔ ④ – ⓒ ⑤ – ⓐ

[문제 8]

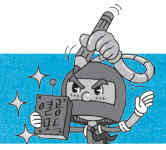
IEEE 1471

[문제 9]

- 장점 : 대량의 데이터를 저장하는데 효과적이고, 컴포넌트의 추가 · 삭제가 편리하며, 중앙 집중화를 통해 데이터 관리가 용이하고, 보안적인 측면이 뛰어나다.
- **단점** : 저장소에 오류가 발생하면 시스템 전체에 문제가 발생하며, 데이터의 분산이 어렵다.

[문제 10]

① 모델(Model) ② 뷰(View) ③ 컨트롤러(Controller)



Section 005

[문제 1]

200, 201, 300

```
#include <stdio.h>
main()
{
  int result, a = 100, b = 200, c = 300;
  result = a < b ? b++ : --c;
  printf("%d, %d, %d\n", result, b, c);
}
```

b++은 후치 연산이고 --c는 전치 연산이므로 a가 b보다 작으면 result에 b의 값 200을 저장한 후 b를 1 증가시키고(b++) 그렇지 않으면 c에서 1을 뺀(--c) 값 299를 저장한다.

결과 200, 201, 300

[문제 2]

2, 1, 1, 1

```
#include <stdio.h>
main()
{
  int hap, j, k, L;
  j = k = L = 0;
  hap = ++j + k++ + ++L;
  printf("%d, %d, %d, %d\n", hap, j, k, L);
}
```

++j와 ++L은 전치 연산이므로 j값과 L값을 1씩 증가시킨 후 연산에 참여하고, k++는 후치 연산이므로 연산에 참여한 후 1을 증가시킨다. 즉 'hap = 1+0+1 = 2' 이후에 k가 1 증가한다.

결과 2, 1, 1, 1

[문제 3]

1, 0

```
#include <stdio.h>
main()
{
  int i = 5, j = 4, k = 1, L, m;
  L = i > 5 || j != 0;
  m = j <= 4 && k < 1;
  printf("%d, %d\n", L, m);
}
```

관계 연산자와 논리 연산자가 있으면 관계 연산자를 먼저 수행한다. 'i > 5'는 거짓이고 'j != 0'은 참이므로 0(거짓) || 1(참) = 1(참)이 되어 L에 1이 저장된다.

'j <= 4'는 참이고 'k < 1'은 거짓이므로 1(참) && 0(거짓) = 0(거짓)이 되어 m에 0이 저장된다.

결과 1, 0

[문제 4]

1, 9, 3

```
#include <stdio.h>
main()
{
  int i = 10, j = 10, k = 30;
  i /= j;        i = i / j와 같다. 즉 i = 10 / 10으로 i에는 1이 저장된다.
  j -= i;        j = j - i와 같다. 즉 j = 10 - 1로 j에는 9가 저장된다.
  k %= j;        k = k % j와 같다. 즉 k = 30 % 9로 k에는 3이 저장된다.
  printf("%d, %d, %d\n", i, j, k);      결과 1, 9, 3
}
```

[문제 5]

소수점a = 1234.6 지수형a = 1.234568e+03

```
#include <stdio.h>
main()
{
  float a = 123456789.0e-5f;        실수형 변수 a를 123456789.0e-5로 초기화한다. 단정도형 실수는 f를 붙여야 한다.
  printf("소수점a = %6.1f  지수형a = %e\n", a, a);
                                    결과 소수점a = 1234.6  지수형a = 1.234568e+03
                                    "소수점a = "을 그대로 출력하고 서식 문자열 "%6.1f"에 대응하는 실수형 변수의 값 123456789.0e-5를 출력한다. 이어서 "지수형a = "을 출력하고 서식 문자열 "%e"에 대응하는 실수형 변수 a의 값을 출력한다. '\n'으로 인해 커서는 다음 줄로 이동한다.
}
```

[문제 6]

1

```
#include <stdio.h>
main()
{
    int a = 5, b = 10, c = 15, d = 30, result;
❶   result = a * 3 + b > d || c - b / a <= d && 1;
    printf("%d\n", result);      결과 1
}
```

❶ 변수에 값을 대입하면 수식은 다음과 같다

```
5 * 3 + 10 > 30 || 15 - 10 / 5 <= 30 && 1;
①     ③    ⑤    ②    ④    ⑥    ⑦    ⑧
```

① 5 * 3 = 15
② 10 / 5 = 2
③ 15 + 10 = 25
④ 15 – 2 = 13
⑤ 25 〉 30 = 거짓
⑥ 13 〈= 30 = 참
⑦ 참(⑥) && 1(참) = 둘 다 참이므로 결과는 참이다.
⑧ 거짓(⑤) || 참(⑦) = 둘 중 하나라도 참이면 참이므로 참(1)이 result에 저장된다.

[문제 7]

20, 24, 36, 80

```
#include 〈stdio.h〉
main( )
{
  int j = 024, k = 24, L = 0x24, hap;
  hap = j + k + L;
  printf("%d, %d, %d, %d\n", j, k, L, hap);
}
```

정수형 변수 j, k, L, hap를 선언하면서 j에는 8진수 24, k에는 10진수 24, L에는 16진수 24를 저장한다.

※ JAVA도 C언어와 마찬가지로 8진수는 숫자 앞에 0을, 16진수는 숫자 앞에 0x를 붙인다.

hap에는 j, k, L의 값이 10진수로 변환되어 더해진(20+24+36) 결과(80)가 저장된다.

- 8진수 24를 10진수로 변환하면 $2 \times 8^1 + 4 \times 8^0 = 20$이다.
- 16진수 24를 10진수로 변환하면 $2 \times 16^1 + 4 \times 16^0 = 36$이다.

결과

```
20, 24, 36, 80
```

Section 006

[문제 1]

36

```
#include 〈stdio.h〉
main()
{
  int i, hap = 0;
  for (i = 1; i 〈 20; i++) ❶
  { ❷
    if (i % 6 == 0) ❸
      hap += i; ❹
  } ❺
  printf("%d \n", hap); ❻
}
```

반복 변수 i가 1에서 시작하여 1씩 증가하면서 20보다 작은 동안 ❷~❺ 사이의 문장을 반복한다. 총 19회 반복

❷~❺까지가 반복문의 범위이다.

i를 6으로 나눈 나머지가 0이면, 즉 6의 배수이면 ❹번 문장을 수행하고, 아니면 ❺번으로 이동한다. 6의 배수만 hap에 누적시키려는 것이다.

'hap = hap + i;'와 동일하다. i의 값, 즉 6의 배수가 hap에 누적된다.

반복문의 끝. 제어가 ❶번으로 이동하여 조건을 판단한 다음 반복 여부를 결정한다.

결과

```
36
```

[문제 2]

6

```
#include <stdio.h>
main()
{
  int i = 0, hap = 0;
  while (1) ❶
  { ❷
    i += 2; ❸
    if (hap > 5) ❹
      break; ❺
    else if (i % 4 == 0) ❻
      hap -= i; ❼
    else ❽
      hap += i; ❾
  } ❿
  printf("%d \n", hap); ⓫
}
```

while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 { } 사이의 문장을 무한 반복한다.
❷~❿까지가 반복문의 범위이다.
'i = i + 2;'와 동일하다. i의 값을 2씩 누적시킨다.
hap이 5보다 크면 ❺번 문장으로 수행하고, 아니면 ❻번으로 이동한다.
반복문을 탈출하여 ⓫번으로 이동한다.
i를 4로 나눈 나머지가 0이면, 즉 i가 4의 배수이면 ❼번을 수행하고, 아니면 ❽번으로 이동한다.
'hap = hap - i;'와 동일하다. i의 값을 hap에서 뺀 다음 ❿번으로 이동한다.
❻번의 조건식이 거짓일 경우 실행할 문장의 시작점이다.
'hap = hap + i;'와 동일하다. i의 값을 hap에 누적시킨 다음 ❿번으로 이동한다.
반복문의 끝. 제어가 ❶번으로 이동하여 조건을 판단한 다음 반복 여부를 결정한다.
결과 6

디버깅

i	hap	출력
0	0	6
2	2	
4	-2	
6	4	
8	-4	
10	6	
12		

[문제 3]

A등급

```
#include <stdio.h>
main()
{
  int jum = 95;
  char lev;
  switch (jum / 30) ❶
  { ❷
  case 3: ❸
    lev = 'A'; ❹
    break; ❺
  case 2: ❻
    lev = 'B'; ❼
    break; ❽
```

jum을 30으로 나눠 결과에 해당하는 숫자를 찾아간다. 95/30은 3.166…이지만 정수 나눗셈은 결과도 정수이므로 결과는 3이다. 3에 해당하는 ❸번으로 이동하여 ❹, ❺번을 실행한다.
❷~⓯까지가 스위치 조건문의 범위이다.
'jum/30'이 3일 경우 찾아오는 곳이다. ❹, ❺번을 실행한다.
lev에 'A'를 저장한다.
switch문을 탈출하여 ⓭번으로 이동한다.
'jum/30'이 2일 경우 찾아오는 곳이다. ❼, ❽번을 실행한다.
lev에 'B'를 저장한다.
switch문을 탈출하여 ⓭번으로 이동한다.

```
  case 1: ❾                  'jum/30'이 1일 경우 찾아오는 곳이다. ❿, ⓫번을 실행한다.
    lev = 'C'; ❿             lev에 'C'를 저장한다.
    break; ⓫                 switch문을 탈출하여 ⓰번으로 이동한다.
  default: ⓬                 case 3~1에 해당되지 않는 경우, 즉 jum이 29이하인 경우 찾아오는 곳이다.
    lev = 'D'; ⓭             lev에 'D'를 저장한다.
    break; ⓮                 switch문을 탈출하여 ⓰번으로 이동한다. 여기에 break가 없어도 switch문의 마지막 문장이므로 switch문
                             을 벗어나 ⓰번으로 이동한다.
  } ⓯                        switch문의 끝
  printf("%c등급\n", lev); ⓰   결과 A등급
}
```

[문제 4]

12

```
#include <stdio.h>
main()
{
  int i = 10, hap = 0;
  while (i > 1) ❶            i가 1보다 큰 동안 ❷~❻ 사이의 문장을 반복하여 수행한다.
  { ❷                        ❷~❻까지가 반복문의 범위이다.
    i--; ❸                   'i = i - 1;'과 동일하다. i의 값을 1씩 감소시킨다.
    if (i % 3 == 1) ❹        i를 3으로 나눈 나머지가 1이면 ❺번 문장을 수행하고, 아니면 ❻번 문장으로 이동한다.
      hap += i; ❺            'hap = hap + i;'와 동일하다. i의 값을 hap에 누적시킨다.
  } ❻                        반복문의 끝. ❶번으로 이동하여 조건을 판단한 다음 반복 여부를 결정한다.
  printf("%d\n", hap); ❼     결과 12
}
```

디버깅

i	hap	출력
10	0	12
9	7	
8	11	
7	12	
6		
5		
4		
3		
2		
1		

Section **007**

[문제 1]

KOR

```
#include <stdio.h>
main()
{
  char *str;
  str = "KOREA";
  printf("%8.3s\n", str);
}
```

문자형 변수가 저장된 곳의 주소를 기억할 포인터 변수 str을 선언한다.

str은 주소를 기억하는 포인트 변수이므로 str에 "KOREA"가 기억되는 것이 아니라 "KOREA"라는 문자열이 메모리의 어딘가에 저장된 후 그 저장된 곳의 주소가 str에 기억된다.

결과 KOR

%s는 문자열을 출력하는 서식 문자열이고 %8.3s는 8자리를 확보하여 출력할 문자열의 왼쪽에서부터 3글자만 출력하라는 의미이다. 그러므로 str이 가리키는 곳의 값인 문자열 "KOREA" 중 앞의 3글자만 출력된다.

※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

[문제 2]

C, A

```
#include <stdio.h>

main()
{
  char a[] = { 'A', 'B', 'C', 'D', 'E', 'F' };
  char *p;
  p = &a[2];
  printf("%c, %c\n", *p, *(p - 2));
}
```

배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

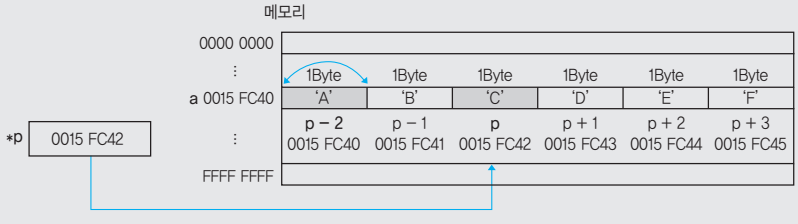
	첫 번째	두 번째	세 번째	네 번째	다섯 번째	여섯 번째
배열 a	A	B	C	D	E	F
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

문자형 변수가 저장된 곳의 주소를 기억할 포인터 변수 p를 선언한다.

&a[2], 즉 배열 a의 세 번째 요소의 주소를 포인터 변수에 기억시킨다. p에는 a[2]의 주소가 기억된다.

p가 가리키는 곳의 값(*p)과 p가 가리키는 곳의 주소에서 2를 뺀(p–2), 즉 주소를 2 번지(p가 문자형 포인터이므로 1Byte씩) 감소시킨 곳의 값(*(p–2))을 출력한다.

p에 저장된 a[2]의 주소와 그 주소에서 2 번지, 즉 2Byte 감소시킨 위치를 그림으로 표현해 보면 다음과 같다.



※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

[문제 3]

1
1
2
4
8

```
#include <stdio.h>
main()
{
  int numAry[] = { 1,0,0,0,0 };  ❶
  int i, j;  ❷
  for (j = 0; j < 5; ++j)  ❸
    for (i = 0; i < j; ++i)  ❹
      numAry[j] += numAry[i];  ❺
  for (j = 0; j < 5; ++j)  ❻
    printf("%d\n", numAry[j]);  ❼
}
```

❶ 배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

정수형 변수 i, j를 선언한다.

❸ 반복 변수 j가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❹번을 반복하여 수행한다. 결국 ❹번 문장을 5회 반복한다.

❹ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 j보다 작은 동안 ❺번을 반복하여 수행한다.

- i는 j보다 작을 때까지 반복해야 하는데, j가 0일 때 i도 0이므로 i가 j보다 작지 않아 한 번도 수행하지 않음
- j가 1일 때 i는 0에서 0까지 1회 반복
- j가 2일 때 i는 0에서 1까지 2회 반복
- j가 3일 때 i는 0에서 2까지 3회 반복
- j가 4일 때 i는 0에서 3까지 4회 반복 수행하므로 ❺번은 총 10회 수행된다.

❺ 'numAry[j] = numAry[j] + numAry[i];'와 같다. numAry[i]를 numAry[j]에 누적한다.

❻ 반복 변수 j가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❼번을 반복하여 수행한다.

❼ numAry[0]~numAry[4]까지 출력한다. 서식 문자열에 '\n'이 포함되어 있으므로 각각의 요소가 서로 다른 줄에 출력된다.

디버깅

j	i	numAry
0	0	1 0 0 0 0
1	0	1 1 0 0 0
	1	
2	0	1 1 1 0 0
2	1	1 1 2 0 0
	2	
3	0	1 1 2 1 0
3	1	1 1 2 2 0
3	2	1 1 2 4 0
	3	
4	0	1 1 2 4 1
4	1	1 1 2 4 2
4	2	1 1 2 4 4
4	3	1 1 2 4 8
	4	
5		

[문제 4]

!moT ma I

```
#include <stdio.h>
#include <string.h>
main()
{
  int k, n;  ❶
  char st[] = "I am Tom!";  ❷

  char temp;  ❸
  n = strlen(st);  ❹

  n--;  ❺

  for (k = 0; k < n; k++)  ❻

  {
    temp = *(st + k);  ❼

    *(st + k) = *(st + n);  ❽
    *(st + n) = temp;  ❾

    n--;  ❿
  }
  printf("%s\n", st);  ⓫
}
```

정수형 변수 k, n을 선언한다.

배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언되는데, 초기값이 문자열인 경우 널 문자가 들어가기 때문에 문자열의 크기보다 1 큰 배열이 만들어진다.

문자형 변수 temp를 선언한다.

strlen()은 문자열의 길이를 구하는 함수이다. st가 가리키는 문자열, 즉 "I am Tom!"의 길이 9를 저장한다. strlen() 함수는 널 문자를 제외한 순수한 문자열의 길이만 구한다.

n의 값을 1 감소시킨다. C 언어에서는 배열의 위치가 0부터 시작하므로 배열 st는 st[0]~st[8]까지 9개의 문자를 저장하게 된다. 즉 교환할 문자의 끝 위치는 8번째이므로 n을 1감소시키는 것이다.

반복 변수 k가 0에서 시작하여 1씩 증가하면서 n보다 작은 동안 ❼~❿번을 반복하여 수행하는데, ❿번에서 n이 1씩 감소하면서 문자의 끝 위치를 하나씩 앞으로 당긴다. 즉 ❼~❿번을 4회 반복한다.

temp에 배열 st가 가리키는 곳의 주소에서 k만큼 증가한(st가 문자형이므로 1Byte씩 증가) 주소가 가리키는 곳의 값(*(st+k))을 저장한다.

(st+k)가 가리키는 곳의 값을 (st+n)이 가리키는 곳의 값으로 치환한다.

(st+n)이 가리키는 곳의 값을 temp에 저장된 값으로 치환한다.

※ 결국 ❼~❾번은 temp 변수를 이용하여 (st+k)가 가리키는 곳의 값과 (st+n)이 가리키는 곳의 값을 서로 교환하는 것이다.

k가 0, n이 8일 때, st[0]번과 st[8]번을 교환, k가 1, n이 7일 때, st[1]번과 st[7]번을 교환,
k가 2, n이 6일 때, st[2]번과 st[6]번을 교환, k가 3, n이 5일 때, st[3]번과 st[5]번을 교환한다.
총 4회 교환 작업을 하고 n과 k가 4가 되었을 때 반복문을 벗어난다.

n의 값을 1 감소시킨다.

%s는 문자열을 출력하는 서식 문자열이므로 배열 st에 저장되어 있는 문자열이 출력된다.

※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

디버깅

k	n	Temp	*(st+k)	*(st+n)	st[] st[0] st[1] st[2] st[3] st[4] st[5] st[6] st[7] st[8] st[9]
	9				I a m T o m ! \0
0	8	I	!	I	~~I~~ a m T o m ~~!~~ \0 → ! I
1	7		m		~~!~~ a m T o ~~m~~ I \0 → m
2	6	a	o	a	! m ~~a~~ m T ~~o~~ ! \0 → o a I
3	5	m	T	m	! m o ~~m~~ T ~~I~~ a I \0 → T m
4	4				! m o T m a I \0

Section **008**

[문제 1]

- a => 2, b => 2
- a => 3, b => 2
- a => 4, b => 2

```
#include <stdio.h>
int a = 1; ❶
```
정수형 전역 변수 a를 선언하고, 초기값으로 1을 할당한다. a는 main() 함수 밖에서 선언했기 때문에 이 파일에 속한 모든 함수에서 사용할 수 있다.

```
void increase(); ❷
```
사용할 함수를 선언한다. 리턴값이 없으므로 void를 붙인다.

```
main()
{
  increase(); ❸
  increase(); ❾
  increase(); ⓯
}
```
인수 없이 increase 함수를 호출한다. ❹번으로 이동한다.
인수 없이 increase 함수를 호출한다. ❿번으로 이동한다.
인수 없이 increase 함수를 호출한다. ⓰번으로 이동한다.

```
void increase() ❹❿⓰
{ ❺⓫⓱
  int b = 1; ❻⓬⓲
```
함수의 리턴 값이 없으므로 void를 붙인다.
increase 함수의 범위이다.
정수형 지역 변수 b를 선언하고, 초기값으로 1을 할당한다. 함수 안에서 선언한 변수를 지역 변수라고 한다. 지역 변수는 선언한 함수 또는 블록 내에서만 사용할 수 있다.

```
  printf("a => %d, b => %d\n", ++a, ++b); ❼⓭⓳
```
결과 a => 2, b => 2
a는 ❶번에서 선언하고 1을 할당했기 때문에 거기에 1을 더하므로 2가 됐고, b는 함수에서 선언하고 1을 할당했기 때문에 거기에 1을 더하니 2가 된 것이다.

⓭ 결과 a => 3, b => 2
a는 전역 변수 이므로 ❼번 수행 후 2의 값을 그대로 가지고 있으므로 1을 더해 3이 됐고, b는 지역 변수 이므로 함수에서 다시 선언하고 1을 할당했기 때문에 거기에 1을 더하니 2가 된 것이다.

⓳ 결과 a => 4, b => 2
a는 전역 변수 이므로 ⓭번 수행 후 3의 값을 그대로 가지고 있으므로 1을 더해 4가 됐고, b는 지역 변수 이므로 함수에서 다시 선언하고 1을 할당했기 때문에 거기에 1을 더하니 2가 된 것이다. 함수에서 선언한 지역 변수는 함수를 벗어나면 소멸된다.

```
} ❽⓮⓴
```
❽ 함수를 마치고 ❾번으로 이동한다.
⓮ 함수를 마치고 ⓯번으로 이동한다.
⓴ 함수를 마치고 프로그램을 종료한다.

[문제 2]

a=10, b=20, c=–10

```
#include <stdio.h>
int prnt(int a, int b); ❶

main()
{
  int a, b, c; ❷
  a = 10; ❸
  b = 20; ❹
  c = prnt(a, b); ❺❿
  printf("a=%d, b=%d, c=%d\n", a, b, c); ⓯

}

int prnt(int x, int y) ❻
{ ❼
  int z; ❽
  if (x == y) ❾
    z = x + y; ❿
  else
    z = x – y; ⓫
  return(z); ⓬
} ⓭
```

사용할 함수를 선언하는 곳이다. 리턴값이 있으므로 void를 생략한다.

정수형 변수 a, b, c를 선언한다.

a의 초기값으로 10을 할당한다.

b의 초기값으로 20을 할당한다.

a, b, 즉 10과 20을 인수로 하여 prnt() 함수를 호출한 다음 결과를 c로 받는다. ❻번으로 이동한다.

결과 a=10, b=20, c=–10

a와 b는 초기값 그대로 10과 20을, c는 ⓬에서 z의 값을 돌려받았음으로 –10을 출력한다.

함수의 리턴 값이 있으므로 void를 생략한다. ❺에서 'prnt(a, b)라고 했으므로 정수형 변수 x는 a의 값 10을 받고, 정수형 변수 y는 b의 값 20을 받는다.

prnt() 함수의 범위이다.

정수형 변수 z를 선언한다.

x와 y가 같으면 ❿번 문장을 실행하고 아니면 ⓫번 문장으로 이동한다.

❾번의 조건식이 참일 경우 실행할 문장이다. 참이 아니기 때문에 z에는 알 수 없는 값이 그대로 남아있다.

❾번의 조건식이 거짓일 경우 실행할 문장이다. a와 b는 같지 않으므로 z에는 'x–y'의 결과인 –10이 저장된다.

z가 가지고 있는 값 –10을 ⓮번으로 이동하여 c에 할당한다.

[문제 3]

a=10, b=10, c=55

```
#include <stdio.h>
void prnt(int *a, int *b, int *c); ❶
```

- void : 리턴값이 없으므로 void를 붙인다.
- (int *a, int *b, int *c) : 함수에서 사용할 인수다. 정수형 포인터 변수 3개를 사용한다는 의미이다. 호출하는 곳에서 보내준 인수의 순서와 자료형이 일치해야 한다.

```
main()
{
  int a = 0, b = 10, c = 0; ❷
  prnt(&a, &b, &c); ❸
  printf("a=%d, b=%d, c=%d\n", a, b, c); ⓬
}
```

정수형 변수 a, b, c를 선언하고, 초기값으로 0, 10, 0을 할당한다.

정수형 변수 a, b, c의 주소를 인수로 하여 prnt() 함수를 호출한다. ❹번으로 이동한다.

결과 a=10, b=10, c=55

a, b, c 변수는 모두 값을 돌려받지 않았지만 a, b, c 변수의 주소에 있는 값이 변경되었으므로 변경된 값 10, 10, 55를 출력한다.

```
void prnt(int *x, int *y, int *z) ❹
{ ❺
  while (*x < *y) ❻
  { ❼
    ++*x; ❽
    *z = *z + *x; ❾
  } ❿
} ⓫
```

함수의 리턴 값이 없으므로 void를 붙인다. ❸에서 'prnt(&a, &b, &c)'라고 했으므로 x는 a의 주소, y는 b의 주소, z는 c의 주소를 받는다.

prnt() 함수의 범위이다.

x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작은 동안 ❽~❾번을 반복 수행한다. x가 가리키는 곳의 값이 0이고, y가 가리키는 곳의 값은 10이므로 10회 반복 수행한다.

❼에서 ❿번까지가 반복문의 범위이다.

x가 가리키는 곳의 값을 1 증가시킨다.

z가 가리키는 곳의 값에 x가 가리키는 곳의 값을 더해 다시 z가 가리키는 곳에 저장한다.

반복문의 끝

함수를 마치고 ⓬번으로 돌아간다.

※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

디버깅

a	b	c	*x	*y	*z	출력
0	10	0	0	10	0	a=10, b=10, c=55
			1		1	
			2		3	
			3		6	
			4		10	
			5		15	
			6		21	
			7		28	
			8		36	
			9		45	
			10		55	

[문제 4]

234

모든 C 프로그램은 반드시 main( ) 함수부터 시작해야 한다.

```
int main( ) {
❶   int result;
❷   result = res200( );
    printf("%d\n", result);
}
```

❶ 정수형 변수 result를 선언한다.

❷ 인수 없이 res200 함수를 호출한 다음 돌려받은 값을 result에 저장한다.

```
int res200( ) {
❶   return 200 + res30( );
}
```

❶ 200에 res30 함수를 호출한 다음 돌려받은 값을 더해서 함수를 호출한 곳(main 함수)으로 반환한다.

```
int res30( ) {
❶   return 30 + res10( );
}
```

❶ 30에 res10 함수를 호출한 다음 돌려받은 값을 더해서 함수를 호출한 곳(res200 함수)으로 반환한다.

```
int res10( ) {
❶   return 4;
}
```

❶ 반환값 4를 가지고 res10( ) 함수를 호출했던 res30( ) 함수로 제어를 옮긴다.

```
int res30( ) {
❶   return 30 + res10( );
}
```

❶ 반환값 34(30+4)를 가지고 res30( ) 함수를 호출했던 res200( ) 함수로 제어를 옮긴다.

```
int res200( ) {
❶   return 200 + res30( );
}
```

❶ 반환값 234(200+34)를 가지고 res200( ) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
int main( ) {
    int result;
❶   result = res200( );
❷   printf("%d\n", result);
}
```

❶ res200( ) 함수에서 돌려받은 234를 result에 저장한다.

❷ result의 값 234를 정수형으로 출력한 후 커서를 다음 줄 처음으로 옮긴다.

결과

```
234
```

[문제 5]

1024

```
int main( ) {
❶   printf("%d\n", power(2, 10));
    return 0;
}
```

❶ 2와 10을 인수로 하여 power 함수를 호출한 다음 돌려받은 값을 정수형으로 출력한다. power(2, 10)로 호출한다.

```
int power(int data, int exp) {
❶   int i, result = 1;
❷   for(i = 0; i < exp; i++)
❸       result = result * data;
❹   return result;
}
```

power( ) 함수가 호출될 때 2와 10을 전달받았으므로 data는 2이고 exp는 10이다.

❶ 정수형 변수 i와 result를 선언하고, result의 초기값으로 1을 할당한다.

❷ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 exp보다 작은 동안 ❸번 문장을 반복한다. exp가 10을 가지고 있으므로 총 10회 반복 수행한다.

❸ result * data의 결과를 result에 저장한다.

❷~❸번을 10회 반복하는 과정은 다음과 같다.

i	result	data	exp
0	1 2	2	10
1	4		
2	8		
3	16		
4	32		
5	64		
6	128		
7	256		
8	512		
9	1024		
10			

❹ result의 값이 1024이므로 반환값 1024를 가지고 power(2, 10) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
int main( ) {
❶   printf("%d\n", power(2, 10));
❷   return 0;
}
```

❶ power(2, 10) 함수에서 돌려받은 1024를 정수형으로 출력한 후 커서를 다음 줄 처음으로 옮긴다.

결과

```
1024
```

❷ main( ) 함수에서 리턴 값으로 0을 반환하는 것은 에러 없이 정상적으로 프로그램이 종료되었다는 의미이다.

[문제 6]

5

11

23

47

95

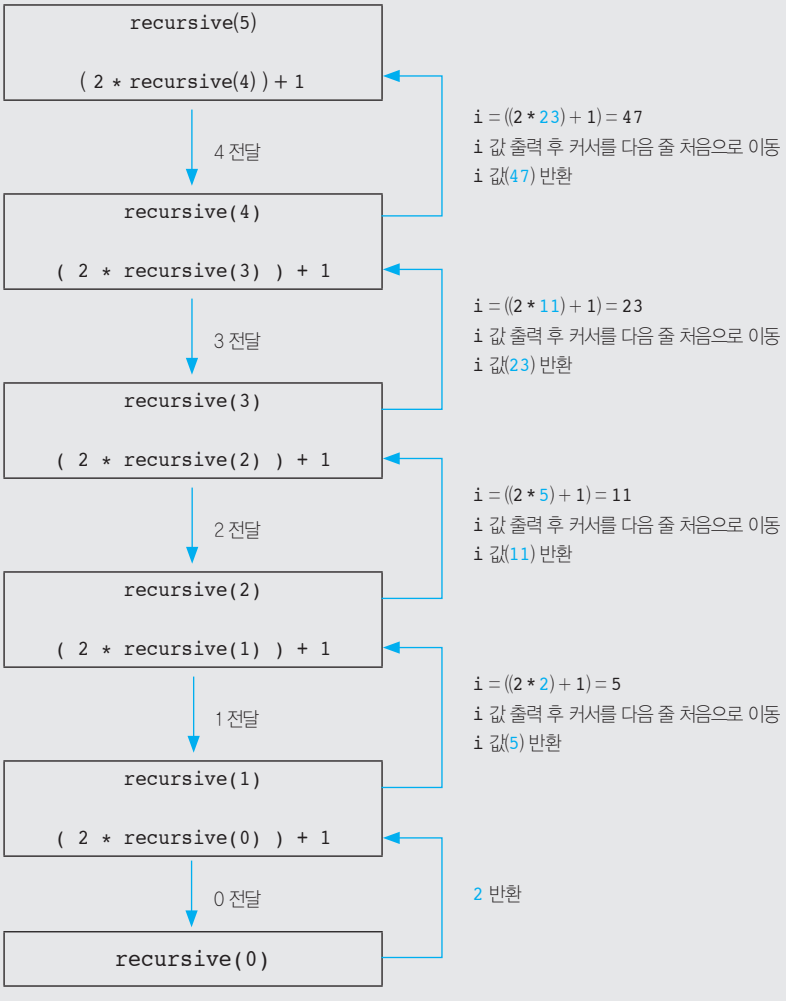
모든 C 프로그램은 반드시 main( ) 함수부터 시작해야 한다.

```
int main(void)
{
❶   int i;
❷   printf("숫자를 입력하시오: ");
❸   scanf("%d", &i);
❹   recursive(i);
}
```

❶ 정수형 변수 i를 선언한다.
❷ "숫자를 입력하시오: "를 출력한다.
❸ 키보드로 숫자를 입력받아 그 값을 i에 저장한다. 5가 입력되었다고 가정하였으므로 i에 5가 저장된다.
❹ i의 값을 인수로 하여 recursive( ) 함수를 호출한다. i가 5이므로 recursive(5)로 호출한다.

```
int recursive(int n)
{
❶   int i;
❷   if (n < 1)
        return 2;
    else
    {
❸       i = (2 * recursive(n - 1)) + 1;
❹       printf("%d\n", i);
❺       return i;
    }
}
```

recursive( ) 함수가 호출될 때 5를 전달받았으므로 n은 5이다.
❶ 정수형 변수 i를 선언한다.
❷의 조건을 만족하지 않으므로 ❸을 수행한다.
❸ (2 * recursive(n-1) + 1)을 수행한 후 결과를 i에 저장한다. 지금부터는 자신이 자신을 호출하는 재귀 함수를 이용한다. n이 1보다 크거나 같을 때까지 자신을 호출하는 과정이 수행되다 n이 0이 될 때 2가 반환되면서 호출했던 과정을 복귀할 때 반환된 값을 이용해 i 값을 계산하여 출력한 후 i 값을 반환한다는 것을 염두에 두고 과정을 개괄적인 그림을 통해 살펴보자.



❸ (2 * recursive(n–1)) + 1 = (2 * 47) + 1의 결과인 95를 i에 저장한다.
❹ i 값 출력 후 커서를 다음 줄 처음으로 이동한다.
❺ 반환값 95를 가지고 recursive(5) 함수를 호출했던 main( ) 함수로 제어를 옮긴 후 main( ) 함수의 종료 브레이스 (})를 만나 프로그램을 종료한다.

[문제 7]

- (A) : 1
- (B) : n-2

[피보나치 수의 정의]

피보나치 수는 0과 1로 시작하며, 다음 피보나치 수는 바로 앞의 두 피보나치 수의 합이 된다.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2574, …

`Fibonacci(n)`

- `if n=0, 0` : n이 0이면 0을 반환
- `if n=1, 1` : n이 1이면 1을 반환
- `others, Fibonacci(n-2) + Fibonacci(n-1)` : 그 외에는 바로 앞의 두 수의 합을 반환

```
int main(void) {
❶   int i=0;
❷   for(i=0; i<10; i++)
❸       printf("%d ", Fibonacci(i));
    return 0;
}
```

모든 C 프로그램은 반드시 `main( )` 함수부터 시작해야 한다.

❶ 정수형 변수 `i`를 선언하고, `i`의 초기값으로 0을 할당한다.

❷ 반복 변수 `i`가 0에서 시작하여 1씩 증가하면서 10보다 작은 동안 ❸번 문장을 반복한다. 즉 ❸번 문장을 10회 반복한다.

❸ `i`의 값을 인수로 하여 `Fibonacci( )` 함수를 호출한 다음 돌려받은 값을 정수형으로 출력한다. 처음에는 `i`가 0이므로 `Fibonacci(0)` 함수를 호출한다.

```
int Fibonacci(int n) {
❶   if ( n == 0 )
❷       return 0;
    else if ( n == 1 )
        return 1;
    else
        return Fibonacci(n-2) + Fibonacci(n-1);
}
```

`Fibonacci( )` 함수가 호출될 때 0을 전달받았으므로 `n`은 0이다. ❶의 조건을 만족하므로 ❷를 수행한다. 반환값 0을 가지고 `Fibonacci(0)` 함수를 호출했던 `main( )` 함수로 제어를 옮긴다.

```
int main(void) {
    int i=0;
❷   for(i=0; i<10; i++)
❶❸      printf("%d ", Fibonacci(i));
    return 0;
}
```

❶ `Fibonacci( )` 함수에서 돌려받은 값 0을 출력한다. 결과 0

❷ `i`의 값을 1증가 시킨 후 최종값과 비교한다. `i`값 1은 10보다 작으므로 ❸번 문장을 수행한다.

❸ `i`의 값을 인수로 하여 `Fibonacci( )` 함수를 호출한 다음 돌려받은 값을 정수형으로 출력한다. `i`가 1이므로 `Fibonacci(1)` 함수를 호출한다.

```
int Fibonacci(int n) {
    if ( n == 0 )
        return 0;
❶   else if ( n == 1 )
❷       return 1;
    else
        return Fibonacci(n-2) + Fibonacci(n-1);
}
```

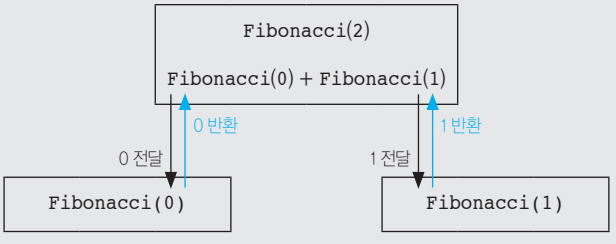
Fibonacci( ) 함수가 호출될 때 1을 전달받았으므로 n은 1이다. ❶의 조건을 만족하므로 ❷를 수행한다. 반환값 1을 가지고 Fibonacci(1) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
int main(void) {
    int i=0;
❷   for(i=0; i<10; i++)
❶❸      printf("%d ", Fibonacci(i));
    return 0;
}
```

❶ Fibonacci( ) 함수에서 돌려받은 값 1을 출력한다. 결과 `0 1`
❷ i의 값을 1증가 시킨 후 최종값과 비교한다. i값 2는 10보다 작으므로 ❸번 문장을 수행한다.
❸ i 값을 인수로 하여 Fibonacci( ) 함수를 호출한 다음 돌려받은 값을 정수형으로 출력한다. i가 2이므로 Fibonacci(2) 함수를 호출한다.
i가 0일 때는 Fibonacci(0) 함수를 호출하여 리턴값이 0이고, i가 1일 때는 Fibonacci(1) 함수를 호출하여 리턴값이 1이라는 것을 이해했다. 나머지 과정은 자신이 자신을 호출하는 재귀 함수를 이용하는 데, 앞에서 이해한 Fibonacci(0), Fibonacci(1)의 결과를 사용하면 된다. 코드는 앞에서 이해한 내용이 반복되므로 이제부터는 두 수의 합으로 피보나치 수가 만들어지는 과정을 개괄적인 그림과 디버깅 표로 설명하겠다.
i=2일 때, Fibonacci(2) 함수를 호출하며 n은 2이다.

```
int Fibonacci(int n) {
    if ( n == 0 )
        return 0;
    else if ( n == 1 )
        return 1;
❶   else
❷       return Fibonacci(n-2) + Fibonacci(n-1);
}
```

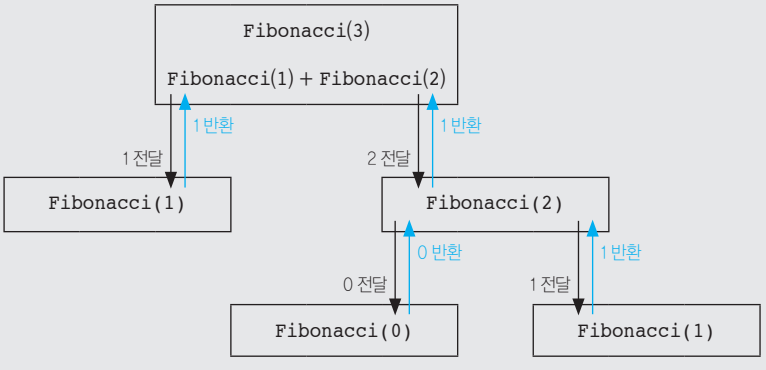
❶의 조건에 해당되므로 ❷번 문장. 즉 'Fibonacci(0) + Fibonacci(1)'을 수행한 후 결과를 반환한다.



- Fibonacci(n-2) + Fibonacci(n-1)은 0 + 1이므로 1을 반환한다.
- main 함수에서는 Fibonacci(2) 함수를 호출하여 1을 돌려받으므로 최종적인 출력 결과는 다음과 같다.

결과 `0 1 1`

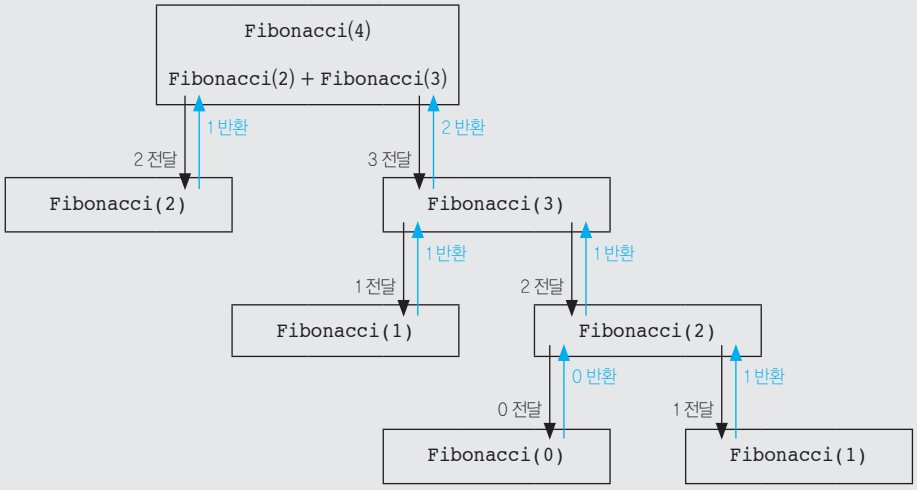
i=3일 때, Fibonacci(3) 함수를 호출하며 n은 3이다.
n이 3이므로 Fibonacci(1) + Fibonacci(2)를 수행한 후 결과를 반환한다.



• Fibonacci(n − 2) + Fibonacci(n − 1)은 1 + 1이므로 2를 반환한다.
• main 함수에서는 Fibonacci(3) 함수를 호출하여 2를 돌려받으므로 최종적인 출력 결과는 다음과 같다.

결과
```
0 1 1 2
```

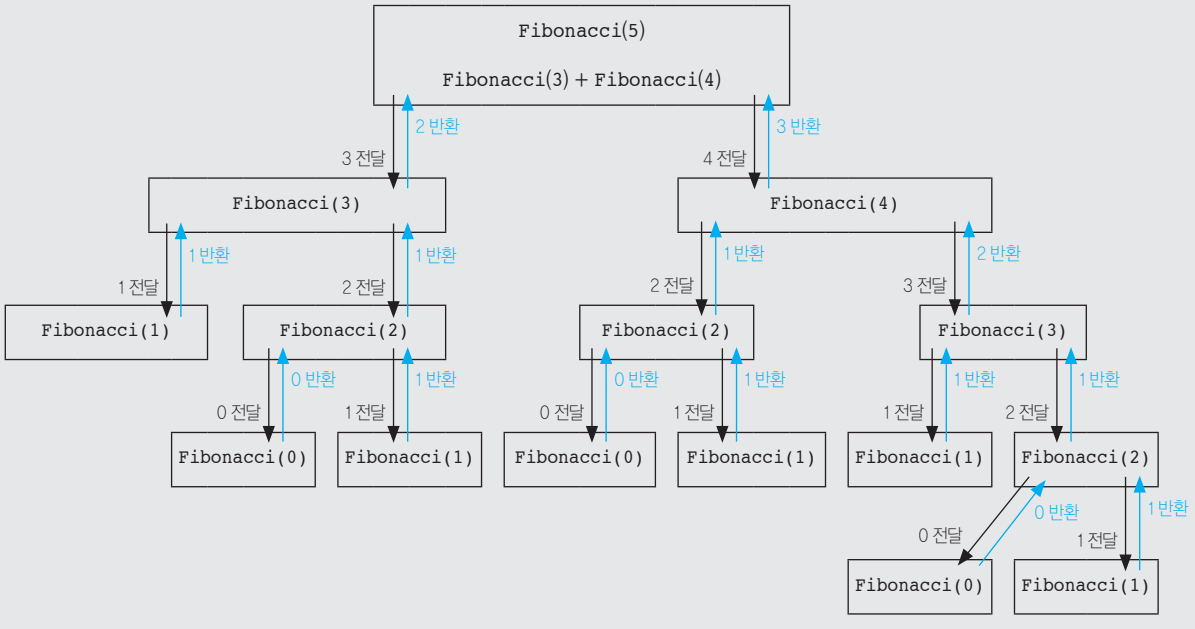
i=4일 때, Fibonacci(4) 함수를 호출하며 n은 4이다.
n이 4이므로 Fibonacci(2) + Fibonacci(3)을 수행한 후 결과를 반환한다.



• Fibonacci(n − 2) + Fibonacci(n − 1)은 1 + 2이므로 3을 반환한다.
• main 함수에서는 Fibonacci(4) 함수를 호출하여 3을 돌려받으므로 최종적인 출력 결과는 다음과 같다.

결과
```
0 1 1 2 3
```

i=5일 때, Fibonacci(5) 함수를 호출하며 n은 5이다.
n이 5이므로 Fibonacci(3) + Fibonacci(4)를 수행한 후 결과를 반환한다.



• `Fibonacci(n − 2) + Fibonacci(n − 1)`은 2 + 3이므로 5를 반환한다.
• `main` 함수에서는 `Fibonacci(5)` 함수를 호출하여 5를 돌려받으므로 최종적인 출력 결과는 다음과 같다.

결과
```
0 1 1 2 3 5
```

이후 과정도 앞에서 살펴본 과정의 반복이다. `main( )` 함수의 `i` 값이 10이 될 때까지 모두 수행해 보자.
위의 과정을 정리하면 디버깅 결과는 다음과 같다.

디버깅

i	함수 호출	n	Fibonacci( ) 함수 리턴값	출력
0	Fibonacci(0)	0	0	0
1	Fibonacci(1)	1	1	1
2	Fibonacci(2)	2	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(0) + Fibonacci(1) = 0 + 1	1
3	Fibonacci(3)	3	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(1) + Fibonacci(2) = 1 + 1	2
4	Fibonacci(4)	4	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(2) + Fibonacci(3) = 1 + 2	3
5	Fibonacci(5)	5	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(3) + Fibonacci(4) = 2 + 3	5
6	Fibonacci(6)	6	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(4) + Fibonacci(5) = 3 + 5	8
7	Fibonacci(7)	7	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(5) + Fibonacci(6) = 5 + 8	13
8	Fibonacci(8)	8	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(6) + Fibonacci(7) = 8 + 13	21
9	Fibonacci(9)	9	Fibonacci(n–2) + Fibonacci(n–1) = Fibonacci(7) + Fibonacci(8) = 13 + 21	34
10				

Section **009**

[문제 1]

3

```
public class Twocheck {
   public static void main(String args[]) {
❶     int[] exint = { 2,4,2,47,6,4,7,2,3,4,5 };

❷     int value = 0;

❸     for(int i = 0; i < exint.length; i++) {

❹        if(exint[i] == 2) {
❺           value++;
         }
      }
❻     System.out.println(value);
   }
}
```

❶ 배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

exint[11]	2	4	2	47	6	4	7	2	3	4	5

❷ 정수형 변수 value를 선언하고 초기값으로 0을 할당한다.

❸ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 exint.length(exint 배열은 11개의 요소를 가지므로 exint.length에는 11이 저장되어 있다.)보다 작은 동안 ❹번을 반복하여 수행한다.

❹ exint 배열의 i번째 값이 2이면 ❺번을 수행한다.

❺ value = value + 1;과 동일하다. value의 값을 1씩 누적시킨다.

❻ 결과 3

디버깅

배열 exint

2	4	2	47	6	4	7	2	3	4	5

value	i	exint[i]	출력
0	0	2	3
1	1	4	
2	2	2	
3	3	47	
	4	6	
	5	4	
	6	7	
	7	2	
	8	3	
	9	4	
	10	5	
	11		

[문제 2]

140

```
public class Problem {
   public static void main(String[] args){
❶    int[][] a = { {11, 12, 13, 14},
                   {21, 22, 23, 24},
                 };
❷    int hap = 0;
❸    for (int i[] : a)
     {
❹      for (int j : i)
❺        hap = hap + j;
     }
❻    System.out.printf("%d", hap);
   }
}
```

❶ 2행 4열의 크기를 갖는 정수형 배열 a가 선언되고 다음과 같이 초기화된다. 중괄호 속의 중괄호는 행을 구분한다. 'int[][] a = { {11, 12, 13, 14}, {21, 22, 23, 24},};'와 같이 한 줄로 입력해도 결과는 같다.

배열 a[2][4]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
	11	12	13	14
	21	22	23	24
	a[1][0]	a[1][1]	a[1][2]	a[1][3]

❸ a 배열의 행 수만큼 ❹번을 반복 수행한다.

- int i[] : a 배열의 한 개의 행이 할당될 변수를 1차원 배열로 선언한다.
- a : 배열의 이름을 적어준다. a 배열이 2행이므로 각 행을 일차원 배열 i[]에 할당하면서 ❹번을 두 번 수행한다.

먼저 1행을 옮기고 ❹번을 4회 반복 수행한다.

배열 i[]	11	12	13	14

이어서 2행을 옮기고 ❹번을 4회 반복 수행한다.

배열 i[]	21	22	23	24

배열 a[2][4]	a[0][0]	a[0][1]	a[0][2]	a[0][3]
	11	12	13	14
	21	22	23	24
	a[1][0]	a[1][1]	a[1][2]	a[1][3]

❹ i 배열의 요소 수만큼 ❺번을 반복 수행한다.

- int j : i 배열의 각 요소가 할당될 변수를 선언한다.
- i : 배열의 이름을 적어준다. i 배열이 4개의 요소를 가지므로 각 요소를 j에 할당하면서 ❺번을 4회 수행한다.

❺ j의 값을 hap에 누적한다.

실행 순서는 다음과 같다.

일차원 배열 i[]가 a 배열의 첫 행을 할당 받았을 때, 즉 ❸번 반복문으로 인해 ❹번 문장을 첫 번째 수행할 때다.

- 첫 번째 수행 : i 배열의 첫 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
11	11	11	12	13	14

- 두 번째 수행 : i 배열의 두 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
23	12	11	12	13	14

- 세 번째 수행 : i 배열의 세 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
36	13	11	12	13	14

• 네 번째 수행 : i 배열의 네 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
50	14	11	12	13	14

일차원 배열 i[]가 a 배열의 두 번째 행을 할당 받았을 때, 즉 ❸번 반복문으로 인해 ❹번 문장을 두 번째 수행할 때다.

• 첫 번째 수행 : i 배열의 첫 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
71	21	21	22	23	24

• 두 번째 수행 : i 배열의 두 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
93	22	21	22	23	24

• 세 번째 수행 : i 배열의 세 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
116	23	21	22	23	24

• 네 번째 수행 : i 배열의 네 번째 값이 j를 거쳐 hap에 누적된다.

hap	j	i 배열			
140	24	21	22	23	24

❻ 결과 140

※ C언어에서는 향상된 for문을 사용할 수 없습니다.

[문제 3]

gnimmargorP

```
public class Problem {
  public static void main(String[] args){
    String str = "Programming";
    int n = str.length();
    char[] st = new char [n];
    n--;
    for (int k = n; k >= 0; k--) {
      st[n-k] = str.charAt(k);
    }
    for (char k : st) {
❶     System.out.printf("%c", k);
    }
  }
}
```

문자열 변수 str의 크기인 11을 정수형 변수 n의 초기값으로 할당한다.
※ length() 메소드는 변수의 크기를 반환한다.

11개의 요소를 갖는 문자 배열 st를 선언한다.

n의 값을 1 감소시킨다.

문자열 변수 str에서 k번째에 있는 문자를 st[n-k]에 저장한다. 결과적으로, 문자열 변수 str의 값을 st[10]~st[0] 순으로 뒤에서부터 차례로 한 글자씩 저장한다.
※ charAt() 메소드는 문자열에서 지정된 위치의 문자를 반환한다.

st 배열의 요소 수만큼 ❶번을 반복 수행한다.
- char k : 문자열 st의 각 요소가 할당될 변수를 선언한다.
- st : 문자 배열의 이름을 적어준다. 문자 배열이 11개의 요소를 가지므로 각 요소를 k에 할당하면서 ❶번을 11회 수행한다.

※ C언어에서는 향상된 for문을 사용할 수 없습니다.

디버깅

n	k	str.charAt(k)	n-k	st[n-k]	배열	출력
11					str[] P r o g r a m m i n g	gnimmargorP
10	10	g	0	g	st[] g	
	9	n	1	n	st[] g n	
	8	i	2	i	st[] g n i	
	7	m	3	m	st[] g n i m	
	6	m	4	m	st[] g n i m m	
	5	a	5	a	st[] g n i m m a	
	4	r	6	r	st[] g n i m m a r	
	3	g	7	g	st[] g n i m m a r g	
	2	o	8	o	st[] g n i m m a r g o	
	1	r	9	r	st[] g n i m m a r g o r	
	0	P	10	P	st[] g n i m m a r g o r P	

[문제 4]

a=–80, b=–90, c=100

```
class IntClass{
  int a,b,c;
}

public class Problem {
  public static void main(String[] args){
        IntClass myVar = new IntClass();
        myVar.a = 10;
        myVar.b = 10;
        myVar.c = 10;
        prnt(myVar);
        System.out.printf("a=%d, b=%d, c=%d\n", myVar.a, myVar.b, myVar.c);
  }

❶ static void prnt(IntClass myVar)

  {
      myVar.a += myVar.b -= myVar.c *= 10;

  }
}
```

IntClass 클래스를 정의한다. class 외부에서 선언했으므로 static을 붙이지 않는다.

객체 변수 myVar을 선언한다.

myVar을 인수로 하여 prnt 함수를 호출한다. ❶번으로 이동한다.

결과 a=–80, b=–90, c=100

실행 클래스 안에서의 정의임으로 static을 붙이고 함수의 리턴 값이 없으므로 void를 붙인다.

계산식의 뒷 부분부터 차례로 계산한다.

myVar.a += myVar.b –= myVar.c *= 10

ⓐ

ⓑ

ⓒ

ⓐ myVar.c *= 10은 myVar.c = myVar.c*10과 같습니다. myVar.c에는 10이 저장되어 있으므로 10*10의 결과값인 100이 myVar.c에 저장됩니다.

ⓑ myVar.b –= myVar.c는 myVar.b = myVar.b – myVar.c와 같습니다. myVar.b에는 10, myVar.c에는 100이 저장되어 있으므로 10 – 100의 결과값인 –90이 myVar.b에 저장됩니다.

ⓒ myVar.a += myVar.b는 myVar.a = myVar.a + myVar.b와 같습니다. myVar.a에는 10, myVar.b에는 –90이 저장되어 있으므로 10+(–90)의 결과값인 –80이 myVar.a에 저장됩니다.

[문제 1]

8, 28

```
public class Problem {
  public static void main(String[] args){
    int i, j = 0;
    for (i = 0; i < 8; i++)
    {
❶     j += i;
    }
    System.out.printf("%d, %d\n", i, j);

  }
}
```

반복 변수 i가 0에서 시작하여 1씩 증가하면서 8보다 작은 동안 ❶번을 8회 반복 수행한다.

'j = j + i;'와 동일하다. i의 값을 j에 누적시킨다.

결과 8, 28

서식 문자열 '%d'에 대응하는 정수 변수 i의 값 8을 출력하고 콤마(,)를 출력한 다음 한 칸 띄고 서식 문자열 '%d'에 대응하는 정수 변수 j의 값 28을 출력한다. '\n'으로 인해 커서는 다음 줄로 이동한다.

[문제 2]

6

```
public class Problem {
  public static void main(String[] args){
    int a = 12, b = 8, c = 2, d = 3;
    a /= b++ - c * d;
    System.out.printf("%d\n", a);
  }
}
```

연산식이 복잡할 때는 우선순위에 맞게 괄호로 묶은 다음 계산하면 쉽다. 연산자의 우선 순위가 같을 때는 좌에서 우로, 대입 연산자는 우선순위가 가장 낮다.

a = a / (b++ - (c * d))
  = 12 / (8 - (2 * 3))
  = 6

※ b++는 후치 연산이므로 연산에 참여한 후 1을 증가시킨다.

결과 6

[문제 3]

–16

28

3

31

```
public class Problem {
  public static void main(String[] args){
    byte a = 15, b = 19;
    System.out.printf("%d\n", ~a);
    System.out.printf("%d\n", a^b);
    System.out.printf("%d\n", a&b);
    System.out.printf("%d\n", a|b);
  }
}
```

결과 –16

- ~ (비트 not)는 각 비트의 부정을 만드는 연산자다. JAVA에서 byte 변수는 정수형 1바이트이므로 각 변수의 값을 1바이트 이진수로 변환한 다음 각 비트에 대해 부정 연산을 한다.

```
15 = 0000 1111
 ~   1111 0000
```

- JAVA는 C와 마찬가지로 2의 보수를 사용하므로 맨 왼쪽 비트는 부호 비트다. 0이면 양수, 1이면 음수이다. 원래의 값을 알기 위해서는 1111 0000에 대한 2의 보수를 구한다. 0001 0000은 10진수로 16이고 원래 음수였으므로 –를 붙이면 –16이다.
- 서식 문자열에 '\n'이 있으므로 결과는 서로 다른 줄에 출력된다.

결과 28

^(비트 xor)는 두 비트가 모두 같으면 0, 서로 다르면 1이 되는 비트 연산자이다.

```
15 = 0000 1111
19 = 0001 0011
 ^   0001 1100
```

0001 1100은 10진수로 28이다.

결과 3

&(비트 and)는 두 비트가 모두 1일 때만 1이 되는 비트 연산자이다.

```
15 = 0000 1111
19 = 0001 0011
 &   0000 0011
```

0000 0011은 10진수로 3이다.

결과 31

| (비트 or)는 두 비트 중 한 비트라도 1이면 1이 되는 비트 연산자이다.

```
15 = 0000 1111
19 = 0001 0011
 |   0001 1111 = 16 + 8 + 4 + 2 + 1 = 31
```

[문제 4]

class

```
public class Problem {
❶   static class Employee{
❷     String name;
❸     int idNum;
❹     int salary;
❺     boolean sex;
    }
    public static void main(String[] args){
❻     Employee myJik = new Employee();
❼     myJik.name = "홍길동";
❽     myJik.idNum = 17001;
❾     myJik.salary = 4500000;
❿     myJik.sex = true;

⓫     System.out.printf("%s\n", myJik.name);
⓬     System.out.printf("%d\n", myJik.idNum);
⓭     System.out.printf("%d\n", myJik.salary);
⓮     System.out.printf("%b\n", myJik.sex);
    }
}
```

클래스를 정의하는 예약어는 class이다.

클래스를 정의하는 예약어는 class이다.

❷~❺ Employee 클래스의 속성(변수)을 정의한다

※ boolean은 참(true)과 거짓(false)을 저장하기 위한 자료형이다.

Employee 클래스의 객체 변수 myJik을 선언한다.

❼~❿ 객체 변수 myJik의 각 속성에 값을 할당한다.

⓫~⓮ 서식 문자열에 '\n'이 있으므로 각각은 서로 다른 줄에 출력된다.

```
홍길동
17001
4500000
true
```

[문제 5]

29

53

35

```
public class Problem {
  public static void main(String[] args){
    int a = 035, b = 0x35, c = 35;

    System.out.printf("%d\n", a);

    System.out.printf("%d\n", b);

    System.out.printf("%d\n", c);
  }
}
```

정수형 변수 a, b, c를 선언하면서 a에는 8진수 35, b에는 16진수 35, c에는 10진수 35를 저장한다.

※ C언어도 JAVA와 마찬가지로 8진수는 숫자 앞에 0을, 16진수는 숫자 앞에 0x를 붙인다.

결과 29

8진수 35를 10진수로 변환하면 $3 \times 8^1 + 5 \times 8^0 = 29$다.

결과 53

16진수 35를 10진수로 변환하면 $3 \times 16^1 + 5 \times 16^0 = 53$이다.

결과 35

[문제 6]

8 2 3 4

```
public class Problem {
  public static void main(String[] args){
    int a, b, c, result;
    a = 1;
    b = 2;
    c = 3;
    result = ++a + b++ + ++c;
    System.out.printf("%d %d %d %d\n", result, a, b, c);
  }
}
```

변수 a에는 1, b에는 2, c에는 3을 저장한다.

++a와 ++c는 전치 연산이므로 a값과 c값을 1씩 증가시킨 후 연산에 참여하고, b++는 후치 연산이므로 연산에 참여한 후 b값을 1 증가시킨다. 즉 'result = 2+2+4 = 8' 이후에 b가 1 증가한다.

결과 8 2 3 4

[문제 7]

20 21 30

```
public class Problem {
  public static void main(String[] args){
    int j, k, L, result;
    j = 10;
    k = 20;
    L = 30;
    result = j < k ? k++ : --L;
    System.out.printf("%d %d %d\n", result, k, L);
  }
}
```

j가 k보다 작으면 k++의 값을 result에 저장하고 그렇지 않으면 --L의 값을 result에 저장한다.

결과 20 21 30

k++은 후치 연산이므로 20을 result에 저장한 후 1 증가한다.

[문제 8]

1234.567871 1.234568e+03

```
public class Problem {
  public static void main(String[] args){
    float a = 123456789.0e-5f;
    System.out.printf("%f  %e\n", a, a);
  }
}
```

실수형 변수 a를 선언하면서 123456789.0e-5를 저장한다. 단정도형 실수는 숫자 뒤에 f를 붙인다.

결과 1234.567871 1.234568e+03

서식 문자열 "%f"에 대응하는 실수형 변수 a의 값을 출력한다. 이어서 서식 문자열의 공백만큼 두 칸을 띄고 서식 문자열 "%e"에 대응하는 실수형 변수 a의 값을 소수점 이상 한 자리만 표시하는 지수 형태로 출력한다. 서식 문자열에 자릿수가 지정되어 있지 않으면 소수 이하는 6자리만 표시한다.

[문제 9]

2

```
public class Problem {
  public static void main(String[] args){
    int a = 2, b = 3, c = 4;
    c = a & b;
    System.out.printf("%d\n", c);
  }
}
```

&(비트 and)는 두 비트가 모두 1일 때만 1이 되는 비트 연산자이다.
JAVA에서 정수형 변수는 4바이트이므로 각 변수의 값을 4바이트 이진수로 변환한 다음 각 비트를 연산한다.

2 = 0000 0000 0000 0000 0000 0000 0000 0010
3 = 0000 0000 0000 0000 0000 0000 0000 0011
& 0000 0000 0000 0000 0000 0000 0000 0010
0000 0000 0000 0000 0000 0000 0000 0010은 10진수로 2다.

결과 2

서식 문자열 '%d'에 대응하는 정수형 변수 c의 값 2를 출력한다.

[문제 10]

5

```
public class Problem {
  public static void main(String[] args){
    int a, b = 10;
    a = 20 % 11 / 3 * 5 - b;
    System.out.printf("%d\n", a);
  }
}
```

연산식이 복잡할 때는 우선순위에 맞게 괄호로 묶은 다음 계산하면 쉽다. 연산자의 우선 순위가 같은 때는 좌에서 우로, 대입 연산자는 우선순위가 가장 낮다.

a = (((20 % 11) / 3) * 5) - 10
= (9 / 3) * 5 - 10
= 15 - 10 = 5

결과 5

[문제 11]

–20

```
public class Problem {
  public static void main(String[] args){
    int a = 10, b = 10, c = 30;
    a -= b -= c;
    a += b;
    String str01 = a <= b ? String.format("%d", a) : String.format("%d", b);
    System.out.printf(str01);
  }
}
```

수식을 풀어쓴 다음 연산자 우선순위에 맞게 계산하면 다음과 같다.

'a -= b -= c;'는 'a = a - (b -= c);'와 같고, 다시 쓰면 'a = a - (b = (b - c));'와 같다.

a = a - (b = (b - c))

ⓐ ⓑ

ⓐ b = 10 - 30 = -20

ⓑ a = 10 - (-20) = 30

'a = a + b;'와 같다. b의 값을 a에 누적시킨다.

a = a + b = 30 + (-20) = 10

a(10)가 b(-20)보다 작거나 같으면 포맷 문자열 '"%d", a'를 str01에 저장하고, 그렇지 않으면 포맷 문자열 '"%d", b'를 str01에 저장한다.

※ String.format() 함수는 지정된 서식과 인수에 대한 문자열을 반환한다.

결과 –20

str01에 '"%d", b'가 저장되어 있으므로 'System.out.printf("%d", b);'를 실행한 것과 같다.

[문제 12]

11, 55

```
public class Problem {
  public static void main(String[] args){
    int i, hap = 0;
    for(i = 1; i <= 10; ++i)
❶     hap += i;
    System.out.printf("%d, %d\n", i, hap);
  }
}
```

반복 변수 i가 1에서 시작하여 1씩 증가하면서 10보다 작거나 같은 동안 ❶번을 반복하여 수행한다.

hap에 i를 누적한다.

결과 11, 55

주의할 점은 반복문을 벗어날 때 반복 변수는 'i<=10'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 10보다 작거나 같은 동안에는 반복문을 수행하고, i가 1 증가하여 11이 되었을 때 반복문을 종료한다.

디버깅

반복 횟수	i	hap
		0
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15
6	6	21
7	7	28
8	8	36
9	9	45
10	10	55
	11	

[문제 13]

3 3 1 1

```
public class Problem {
  public static void main(String[] args){
    int a, b, c, hap;
    a = b = c = 2;
    hap = ++a | b-- & c--;
    System.out.printf("%d %d %d %d", hap, a, b, c);
  }
}
```

변수 a, b, c에 2를 저장한다.

++a는 전치 연산이므로 a값을 1 증가시킨 후 연산에 참여하고, b--와 c--는 후치 연산이므로 연산에 참여한 후 b값과 c값을 각각 1씩 감소시킨다.

비트 연산자의 우선 순위는 &, ^, |이므로 계산 순서는 다음과 같다.

hap = ++a | b-- & c--;

ⓑ ⓐ

ⓐ &(비트 and)는 두 비트가 모두 1일 때만 1이 되는 비트 연산자이다.

자바에서 정수형 변수는 4바이트이므로 각 변수의 값을 4바이트 이진수로 변환한 다음 각 비트를 연산한다.

```
 2 = 0000 0000 0000 0000 0000 0000 0000 0010
 2 = 0000 0000 0000 0000 0000 0000 0000 0010
 &   0000 0000 0000 0000 0000 0000 0000 0010  = 10진수로 2다.
```

ⓑ | (비트 or)는 두 비트 중 한 비트라도 1이면 1이 되는 비트 연산자이다.

```
   3  = 0000 0000 0000 0000 0000 0000 0000 0011
 ⓐ(2) = 0000 0000 0000 0000 0000 0000 0000 0010
   |    0000 0000 0000 0000 0000 0000 0000 0011 = 10진수로 3이다.
```

결과 3 3 1 1

[문제 14]

Info

Info

```
public class Problem {
  public static void main(String[] args){
    String a = "Information";
    System.out.printf("%-10.4s\n", a);
    System.out.printf("%10.4s\n", a);
  }
}
```

문자열 변수 a에 "Information"을 저장한다.
※ String은 문자열 객체 변수를 생성하기 위한 클래스다.

결과 Info

%s는 문자열을 출력하는 서식 문자열이고, %-10.4s는 10자리를 확보하여 출력할 문자열의 왼쪽에서부터 4글자만 출력하라는 의미이다. 10자리를 확보한 다음 a에 저장된 문자열 "Information" 중 앞의 4글자를 왼쪽에서부터 출력하므로 결과는 "Info "이다.

결과 Info

%10.4s는 10자리를 확보하여 출력할 문자열의 왼쪽에서부터 4글자만 출력하라는 의미이다. 10자리를 확보한 다음 a에 저장된 문자열 "Information" 중 앞의 4글자를 오른쪽에서부터 출력하므로 결과는 " Info"이다.

[문제 15]

!noitamrofnI

```
public class Problem {
  public static void main(String[] args){
    String str = "Information!";
    int n = str.length();
    char[] st = new char [n];
    n--;
    for (int k = n; k >= 0; k--) {
      st[n - k] = str.charAt(k);
    }
    for (char c:st) {
❶     System.out.printf("%c", c);
    }
  }
}
```

문자열 변수 str의 크기인 12를 정수형 변수 n의 초기값으로 할당한다.
※ length() 메소드는 문자열 변수에 저장된 문자열의 길이를 반환한다.

12개의 요소를 갖는 문자 변수 st를 선언한다.

n의 값을 1 감소시킨다. 배열의 위치가 0부터 시작하므로 배열 st는 st[0]~st[11]까지 12개의 문자를 저장하게 된다.

문자열 변수 str에서 k번째에 있는 문자를 st[n-k]에 저장한다. 결과적으로, 문자열 변수 str의 값을 st[11]~st[0] 순으로 뒤에서부터 차례로 한 글자씩 저장한다.
※ charAt 함수는 문자열에서 지정된 위치의 문자를 읽어온다.

st 배열의 요소 수만큼 ❶번을 반복 수행한다.
- char c : 문자열 st의 각 요소가 할당될 변수를 선언한다.
- st : 문자열 변수의 이름을 적어준다. 문자열 변수가 12개의 요소를 가지므로 각 요소를 c에 할당하면서 ❶번을 12회 수행한다.

디버깅

n	k	str.charAt(k)	n-k	st[n-k]	배열	출력
12					str[] l n f o r m a t i o n !	!noitamrofnl
11	11	!	0	!	st[] !	
	10	n	1	n	st[] ! n	
	9	o	2	o	st[] ! n o	
	8	i	3	i	st[] ! n o i	
	7	t	4	t	st[] ! n o i t	
	6	a	5	a	st[] ! n o i t a	
	5	m	6	m	st[] ! n o i t a m	
	4	r	7	r	st[] ! n o i t a m r	
	3	o	8	o	st[] ! n o i t a m r o	
	2	f	9	f	st[] ! n o i t a m r o f	
	1	n	10	n	st[] ! n o i t a m r o f n	
	0	l	11	l	st[] ! n o i t a m r o f n l	

[문제 16]

50, true, false

```
public class Problem {
  public static void main(String[] args){
    int a, b;
    boolean c, d;          boolean 변수로 c와 d를 선언한다. boolean 변수는 참(true)과 거짓(false)을 저장한다.
    a = 10; b = 0;
    a *= b = 5;            'a = a * (b = 5);'와 같다. 즉 10*50이므로 50이 a에 저장된다.
    c = (a != b);          a의 값과 b의 값이 같지 않으면 true를, 같으면 false를 c에 저장한다.
    d = (a == b);          a의 값과 b의 값이 같으면 true를, 같지 않으면 false를 d에 저장한다.
    System.out.printf("%d, %b, %b\n", a, c, d);        결과 50, true, false
                                                      ※ boolean 변수는 '%b'로 출력한다.
  }
}
```

[문제 17]

12, 3, 6, 10

```
public class Problem {
  public static void main(String[] args){
    int a = 3, b = 4, c = 5, d = 5;
    a += 6 + --b;          a = a + (6 + --b);와 같다.
                             = 3 + (6 + 3)
                             = 12
                           ※ --b는 전치 연산이므로 b값을 1 감소시킨 후 연산에 참여한다.
    d *= 7 - c++;          d = d * (7 - c++);와 같다.
                             = 5 * (7 - 5)
                             = 10
                           ※ c++는 후치 연산이므로 연산에 참여한 후 c값을 1 증가시킨다.
    System.out.printf("%d, %d, %d, %d\n", a, b, c, d);   결과 12, 3, 6, 10
  }
}
```

[문제 18]

5, 15

```
public class Problem {
  public static void main(String[] args){
❶   int i = 0, hap = 0;
❷   do {                  do~while 반복문의 시작점이다. ❸~❹번 문장을 반복하여 수행한다.
❸     ++i;                    'i = i + 1;'과 동일하다. i의 값을 1씩 누적시킨다.
❹     hap += i;               'hap = hap + i;'와 동일하다. i의 값을 hap에 누적시킨다.
❺   } while(i < 5);           i가 5보다 작은 동안 ❸~❹번 문장을 반복하여 수행한다.
❻   System.out.printf("%d, %d\n", i, hap);   결과 5, 15
                                              i가 5가 되었을 때 5를 hap에 누적한 다음 do~while문을 탈출하기 때문에 i
                                              는 5로 끝난다.
  }
}
```

디버깅

반복 횟수	i	hap
	0	0
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

[문제 19]

11 , 4

```
public class Problem {
   public static void main(String[] args){
      int hap1, hap2;
      hap1 = 10 + 10 % 4 - 10 % 9;
      hap2 = 10 * 10 % 4 - 10 % 9 + 5;
      System.out.printf("%d , %d\n", hap1, hap2);
   }
}
```

hap1에 '10 + (10 % 4) - (10 % 9)'의 결과인 11이 저장된다.

hap2에 '((10 * 10) % 4) - (10 % 9) + 5'의 결과인 4가 저장된다.

결과 11 , 4

[문제 20]

10, 55

```
public class Problem {
   public static void main(String[] args){
      int a, hap = 0 ;
      for(a = 0; a < 10; ++a, hap += a);
      System.out.printf("%d, %d\n", a, hap);
   }
}
```

반복문 for는 for(식1;식2;식3)와 같이 초기값, 최종값, 증가값으로 사용할 식을 '식1', '식2', '식3'에 적는데, 여기서는 증가값을 지정하는 '식3' 자리에 수식 두 개가 콤마(,) 연산자로 나열되어 있다. 이건 나열된 두 식을 차례대로 수행하라는 의미이므로 ++a와 hap += a를 순서대로 수행한 후 '식2'를 확인한다.

결과 10, 55

주의할 점은 반복문을 벗어날 때 반복 변수는 'a<10'의 결과가 거짓이 되도록 증가한 후 빠져나간다는 것이다. 여기서는 a가 10보다 작은 동안에는 반복문을 수행하고, a가 1 증가하여 10이 되었을 때 'hap += a'를 수행한 다음 반복문을 탈출한다는 것이다.

디버깅

반복 횟수	a	hap
		0
1	0	0
2	1	1
3	2	3
4	3	6
5	4	10
6	5	15
7	6	21
8	7	28
9	8	36
10	9	45
	10	55

[문제 21]

Powe

```
public class Problem {
  public static void main(String[] args){
    String str;                          문자열 변수 str을 선언한다.
    str = "Power overwhelming!";         str 변수에 "Power overwhelming!"를 저장한다.
    System.out.printf("%8.4s\n", str);   결과  Powe
                                         %s는 문자열을 출력하는 서식 문자열이고, %8.4s는 8자리를 확보한 다음 str에 저장된 문자열
                                         "Power overwhelming!" 중 앞의 4글자를 오른쪽에서부터 출력하므로 결과는 "    Powe"
                                         이다.
  }
}
```

[문제 22]

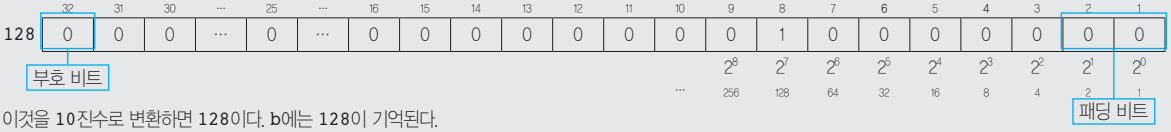
4, 128, 64

```
public class Problem {
  public static void main(String[] args){
❶   int a = 32, b, c;
❷   b = a << 2;
❸   a >>= 3;
❹   c = a << 4;
❺   System.out.printf("%d, %d, %d\n", a, b, c);
  }
}
```

❷ <<는 왼쪽 시프트 연산자이므로, a에 저장된 값을 왼쪽으로 2비트 이동시킨 다음 그 값을 b에 저장시킨다. int는 4Byte이므로 4Byte 2진수로 변환하여 계산하면 된다.

• 4바이트에 32를 2진수로 표현하면 다음과 같다.

• 부호를 제외한 전체 비트를 왼쪽으로 2비트 이동시킨다. 부호는 맨 왼쪽의 비트인데, 0이면 양수, 1이면 음수이므로 빈 자리(패딩 비트)에는 0이 들어오면 된다.



이것을 10진수로 변환하면 128이다. b에는 128이 기억된다.

❸ 'a = a >> 3'과 동일하며 >>는 오른쪽 시프트 연산자이므로, a에 저장된 값을 오른쪽으로 3비트 이동시킨 다음 그 값을 다시 a에 저장시킨다. int는 4Byte이므로 4Byte 2진수로 변환하여 계산하면 된다.

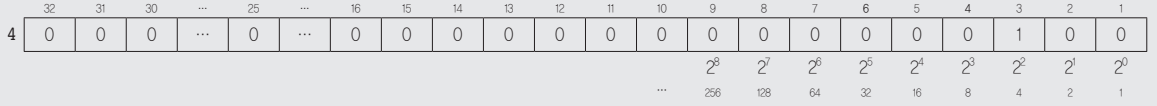
• 4바이트에 32를 2진수로 표현하면 다음과 같다.

• 부호를 제외한 전체 비트를 오른쪽으로 3비트 이동시킨다. 부호는 맨 왼쪽의 0인데, 수치가 작아서 큰 의미가 없다. 양수이므로 빈 자리(패딩 비트)에는 0이 들어오면 된다.

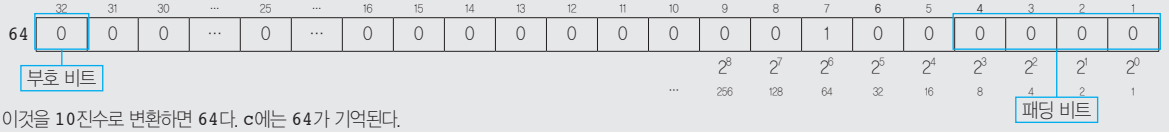
이것을 10진수로 변환하면 4다. a에는 4가 기억된다.

❹ 〈〈는 왼쪽 시프트 연산자이므로, a에 저장된 값 4를 왼쪽으로 4비트 이동시킨 다음 그 값을 c에 저장시킨다.

• 4바이트에 4를 2진수로 표현하면 다음과 같다.



• 부호를 제외한 전체 비트를 왼쪽으로 4비트 이동시킨다. 양수이므로 빈 자리(패딩 비트)에는 0이 들어오면 된다.



이것을 10진수로 변환하면 64다. c에는 64가 기억된다.

❺ 결과 4, 128, 64

[문제 23]

2, 3

3, 3

```
public class Problem {
  public static int a = 1; ❶
  public static void main(String[] args){
    increase(); ❷
    increase(); ❻
  }

  static void increase(){ ❸❼
    int b = 2; ❹❽
    System.out.printf("%d, %d\n", ++a, ++b); ❺❾
  }
}
```

정수형 전역 변수 a를 선언하고, 초기값으로 1을 할당한다. a는 main() 함수 밖에서 선언했기 때문에 이 파일에 속한 모든 함수에서 사용할 수 있다.

인수 없이 increase 함수를 호출한다. ❸번으로 이동한다.

인수 없이 increase 함수를 호출한다. ❼번으로 이동한다.

함수의 리턴 값이 없으므로 void를 붙인다.

정수형 지역 변수 b를 선언하고, 초기값으로 2를 할당한다. 함수 안에서 선언한 변수를 지역 변수라 한다. 지역 변수는 선언한 함수 또는 블록 내에서만 사용할 수 있다.

❺ 결과 2, 3

a는 ❶번에서 선언하고 1을 할당했기 때문에 거기에 1을 더하므로 2가 됐고, b는 함수에서 선언하고 2를 할당했기 때문에 거기에 1을 더하니 3이 된 것이다. 함수를 마치고 ❻번으로 이동한다.

❾ 결과 3, 3

a는 전역 변수이므로 ❷번 수행 후 2를 그대로 가지고 있으므로 1을 더해 3이 됐고, b는 지역 변수이므로 함수에서 다시 선언하고 2를 할당했기 때문에 거기에 1을 더하니 3이 된 것이다. 함수에서 선언한 지역 변수는 함수를 벗어나면 소멸된다. 함수를 마치고 프로그램을 종료한다.

[문제 24]

0 2

```
public class Problem {
  public static void main(String[] args){
    int a = 2, b = 3, c = 4, d, e;
    d = a & b & ~b;
    e = a | b & c;
    System.out.printf("%d %d", d, e);
  }
}
```

정수형 변수 a, b, c, d, e를 선언하면서 a에는 2, b에는 3, c에는 4를 저장한다.

비트 연산자의 우선 순위는 ~, &, |이므로 계산 순서는 다음과 같다.

d = a & b & ~b

ⓑ ⓐ

ⓒ

ⓐ ~ (비트 not)는 각 비트의 부정을 만드는 연산자다.

3 = 0000 0000 0000 0000 0000 0000 0000 0011

~ 1111 1111 1111 1111 1111 1111 1111 1100

ⓑ &(비트 and)는 두 비트가 모두 1일 때만 1이 되는 비트 연산자이다.

JAVA에서 정수형 변수는 4바이트이므로 각 변수의 값을 4바이트 2진수로 변환한 다음 각 비트를 연산한다.

2 = 0000 0000 0000 0000 0000 0000 0000 0010

3 = 0000 0000 0000 0000 0000 0000 0000 0011

& 0000 0000 0000 0000 0000 0000 0000 0010 = 10진수로 2다.

ⓒ ⓑ와 ⓐ의 결과를 & 연산한다.

ⓐ (~3) = 1111 1111 1111 1111 1111 1111 1111 1100

ⓑ (2) = 0000 0000 0000 0000 0000 0000 0000 0010

& 0000 0000 0000 0000 0000 0000 0000 0000 = 10진수로 0이다.

비트 연산자의 우선 순위는 ~, &, |이므로 계산 순서는 다음과 같다.

e = a | b & c

ⓐ

ⓑ

ⓐ &(비트 and)는 두 비트가 모두 1일 때만 1이 되는 비트 연산자이다.

3 = 0000 0000 0000 0000 0000 0000 0000 0011

4 = 0000 0000 0000 0000 0000 0000 0000 0100

& 0000 0000 0000 0000 0000 0000 0000 0000 = 10진수로 0이다.

ⓑ | (비트 or)는 두 비트 중 한 비트라도 1이면 1이 되는 비트 연산자이다.

2 = 0000 0000 0000 0000 0000 0000 0000 0010

ⓐ(0) = 0000 0000 0000 0000 0000 0000 0000 0000

| 0000 0000 0000 0000 0000 0000 0000 0010 = 10진수로 2이다.

결과

```
0 2
```

[문제 25]

11, 65

```
public class Problem {
   public static void main(String[] args){
      int i, hap = 0;
      for(i = 1; i <= 10; ++i, hap += i);
      System.out.printf("%d, %d\n", i, hap);
   }
}
```

반복문 for는 for(식1; 식2; 식3)와 같이 초기값, 최종값, 증가값으로 사용할 식을 '식1', '식2', '식3'에 적는데, 여기서는 증가값을 지정하는 '식3' 자리에 수식 두 개가 콤마(,) 연산자로 나열되어 있다. 이건 나열된 두 식을 차례대로 수행하라는 의미이므로 ++i와 hap += i를 순서대로 수행한 후 '식2'를 확인한다.

결과 11, 65

주의할 점은 반복문을 벗어날 때 반복 변수는 'i<=10'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 10보다 작거나 같은 동안에는 반복문을 수행하고, i가 1 증가하여 11이 되었을 때 'hap += i'를 수행한 후 반복문을 탈출한다는 것이다.

디버깅

반복 횟수	i	hap
		0
1	1	0
2	2	2
3	3	5
4	4	9
5	5	14
6	6	20
7	7	27
8	8	35
9	9	44
10	10	54
	11	65

[문제 26]

true

```
public class Problem {
  public static void main(String[] args){
    int i = 0, hap = 0;
    while(true){                          조건을 만족하는 동안 반복하는 것인데, 조건이 true, 즉 참이므로 무한 반복한다. 결국 ❸번의 조건을 만족하여 break를 만나기 전까지 ❶~❸ 사이의 문장을 반복하여 수행한다.
❶     ++i;                              'i = i + 1;'과 동일하다. i의 값을 1씩 누적시킨다.
❷     hap += i;                         'hap = hap + i;'와 동일하다. i의 값을 hap에 누적시킨다.
❸     if(i >= 100) break;               i가 100보다 크거나 같으면 반복문(while)을 빠져나온다. 제어가 ❹번으로 이동된다.
    }
❹   System.out.printf("1에서 100까지의 합은 %d입니다.\n", hap);     결과 1에서 100까지의 합은 5050입니다.
  }
}
```

디버깅

i	hap
0	0
1	1
2	3
3	6
4	10
5	15
⋮	⋮
96	4656
97	4753
98	4851
99	4950
100	5050

[문제 27]

i=20, j=10

```
public class Problem {
❶ static int i, j;                  정수형 전역 변수 i, j를 선언한다. main() 함수 밖에서 선언했기 때문에 이 파일에서는 어디서나 사용할 수 있
                                    고, 프로그램이 종료할 때까지 값을 유지한다.
  public static void main(String[] args){
❷   i = 10;                        i의 초기값으로 10을 할당한다.
❸   j = 20;                        j의 초기값으로 20을 할당한다.
❹   change();                      인수 없이 change() 함수를 호출한다. ❺번으로 이동한다.
⓫   System.out.printf("i=%d, j=%d\n", i, j);    결과 i=20, j=10
  }

❺ static void change(){            함수의 리턴 값이 없으므로 void를 붙인다.
❻   int temp;                      정수형 지역 변수 temp를 선언한다.
❼   temp = i;                      ❼~❾ i의 값과 j의 값을 교환한다.
❽   i = j;
❾   j = temp;
⓾ }             change() 함수의 끝이다. 제어가 ⓫번으로 이동된다.
                함수의 리턴값을 돌려주지 않지만 i, j가 전역 변수이기 때문에 변경된 값이 적용된다.
}
```

[문제 28]

120.000

```
#include <stdio.h>

main() {
    char c[4] = { '+', '-', '/', '*' };
    double p;
    switch (c[3])
    {
    case '+':
      p = 0;
      for (int i = 0; i < 5; i++, p += i);
      break;
    case '-':
      p = 0;
      for (int i = 0; i < 5; i++, p -= i);
      break;
    case '/':
      p = 1;
      for (int i = 0; i < 5; i++, p /= i);
      break;
❶   default:
      p = 1;
      for (int i = 0; i < 5; i++, p *= i);

    }
    printf("%.3f", p);
}
```

c[3]에 해당하는 문자를 찾아간다. 배열은 0부터 시작하므로 4번째인 '*'을 case에서 찾는다. 없으므로 ❶번으로 이동한다.

case '+', '-', '/'에 해당되지 않는 경우 찾아오는 곳이다.

p에 1을 저장한다.

반복문 for는 for(식1; 식2; 식3)와 같이 초기값, 최종값, 증가값으로 사용할 식을 '식1', '식2', '식3'에 적는데, 여기서는 증가값을 지정하는 '식3' 자리에 수식 두 개가 콤마(,) 연산자로 나열되어 있다. 이건 나열된 두 식을 차례대로 수행하라는 의미이므로 'i++'와 'p *= i'를 순서대로 수행한 후 '식2'를 확인한다.

switch문의 끝

결과 120.000

서식 문자열 "%.3f"에 대응하는 p의 값 120을 소수점 이하 3자리를 잡아 출력하므로 결과는 120.000이다.

주의할 점은 반복문을 벗어날 때 반복 변수는 'i < 5'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 5보다 작은 동안에는 반복문을 수행하고, i가 1 증가하여 5가 되었을 때 'p *= i'를 처리한 다음 종료한다.

디버깅

i	p
0	1
1	1
2	2
3	6
4	24
5	120

[문제 29]

5, –5

```
public class Problem {
  public static void main(String[] args){
    int a, b = 10;
    for(a = 0; a < 5; ++a, b -= a);

      System.out.printf("%d, %d\n", a, b);

  }
}
```

반복문 for는 for(식1; 식2; 식3)와 같이 초기값, 최종값, 증가값으로 사용할 식을 '식1', '식2', '식3'에 적는데, 여기서는 증가값을 지정하는 '식3' 자리에 수식 두 개가 콤마(,) 연산자로 나열되어 있다. 이건 나열된 두 식을 차례대로 수행하라는 의미이므로 '++a'와 'b -= a'를 순서대로 수행한 후 '식2'를 확인한다.

결과 5, –5

주의할 점은 반복문을 벗어날 때 반복 변수는 'a<5'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 a가 5보다 작은 동안에는 반복문을 수행하고, a가 1 증가하여 5가 되었을 때 'b -= a'를 수행한 다음 반복문을 탈출한다는 것이다.

디버깅

a	b
0	10
1	9
2	7
3	4
4	0
5	–5

[문제 30]

inNum % 2

```
import java.util.Scanner;

public class Problem {
  public static void main(String[] args){
    int inNum;
❶   Scanner scan01 = new Scanner(System.in);

❷   inNum = scan01.nextInt();

❸   if((inNum % 2) == 0)
❹     System.out.printf("%d= 짝수입니다.\n", inNum);

❺   else
❻     System.out.printf("%d= 홀수입니다.\n", inNum);

    scan01.close();

  }
}
```

Scanner() 함수가 정의되어 있는 헤더 파일이다.

❶ Scanner 클래스의 객체 변수 scan01을 키보드로 입력받을 수 있도록 생성한다. System.in은 표준 입력장치, 즉 키보드를 의미한다.

❷ 키보드로부터 정수형 값을 입력받아 inNum에 저장한다.
❶, ❷번은 JAVA에서 키보드로 자료를 입력받을 때는 이렇게 하는구나! 정도로만 알아두세요.

❸ inNum을 2로 나눈 나머지가 0이면 ❹번을 실행하고, 아니면 ❺번의 다음 문장인 ❻번을 실행한다.

❹ 결과(입력한 값이 2이라면) 2= 짝수입니다.

❺ ❸번의 조건이 거짓일 경우 실행할 문장의 시작점이다.

❻ 결과(입력한 값이 3이라면) 3= 홀수입니다.

Scanner 클래스의 객체 변수는 임의의 메모리 영역을 확보하여 사용하는 것이므로 프로그램 종료 전에 사용하던 메모리 영역을 해제해야 다른 프로그램이 해당 영역을 사용할 수 있다.

[문제 31]

합은 55입니다.

```
public class Problem {
  public static void main(String[] args){
    int i = 0, hap = 0;
    do{                          do~while 반복문의 시작점이다. ❶~❷번 문장을 반복하여 수행한다.
❶     ++i;
❷     hap += i;
❸   } while(i < 10);             i가 10보다 작은 동안 ❶~❷번 문장을 반복 수행한다.
    System.out.printf("합은 %5d입니다.\n", hap);
  }
}
```

결과 합은 55입니다.

i가 10이 되었을 때 10을 hap에 누적한 다음 do~while문을 탈출하기 때문에 i는 10으로 끝난다.

디버깅

i	hap
0	0
1	1
2	3
3	6
4	10
5	15
6	21
7	28
8	36
9	45
10	55

[문제 32]

A 65

```
public class Problem {
  public static void main(String[] args){
    int ch = 'A';              정수형 변수 ch를 선언하고, 초기값으로 'A'를 할당한다. 'A'의 아스키코드 값이 저장된다.
    System.out.printf("%c %d\n", ch, ch);
  }
}
```

결과 A 65

ch의 값을 %c로 출력하면 'A'이고, %d로 출력하면 65이다.

[문제 33]

B, C

```
public class Problem {
   static class CharClass{
      char a[] = { 'A','B','C','D','E','F' };

   }
   public static void main(String[] args){
      CharClass myVar = new CharClass();
      System.out.printf("%c, %c\n", myVar.a[1], myVar.a[2]);

   }
}
```

CharClass 클래스를 정의한다.

배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

배열 a	A	B	C	D	E	F
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]

CharClass 객체 변수 myVar을 선언한다.

결과 `B, C`

myVar.a[1]의 값과 myVar.a[2]의 값을 출력한다.

[문제 34]

합은1 0입니다.
합은2 2입니다.
합은3 5입니다.
합은4 9입니다.
합은5 14입니다.

```
public class Problem {
   public static void main(String[] args){
      int i = 0, hap = 0;
      for(i = 1; i <= 5; ++i, hap + =i)

         System.out.printf("합은%d  %4d입니다.\n", i, hap);

   }
}
```

반복문 for는 for(식1; 식2; 식3)와 같이 초기값, 최종값, 증가값으로 사용할 식을 '식1', '식2', '식3'에 적는데, 여기서는 증가값을 지정하는 '식3' 자리에 수식 두 개가 콤마(,) 연산자로 나열되어 있다. 이건 나열된 두 식을 차례대로 수행하라는 의미이므로 ++i와 hap += i를 순서대로 수행한 후 '식2'를 확인한다.

주의할 점은 반복문을 벗어날 때 반복 변수는 'i<=5'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 5보다 작거나 같은 동안에는 반복문을 수행하고, i가 1 증가하여 6이 되었을 때 'hap += i'를 수행한 다음 반복문을 탈출한다.

디버깅

반복 횟수	i	hap	출력
	0	0	
1	1	0	합은1 0입니다.
2	2	2	합은2 2입니다.
3	3	5	합은3 5입니다.
4	4	9	합은4 9입니다.
5	5	14	합은5 14입니다.
	6	20	

[문제 35]

gnoganiS

```
public class Problem {
  public static void main(String[] args){
❶   String str = "Sinagong";
❷   int n = str.length();
❸   char[] st = new char [n];
❹   n--;
❺   for (int k = 0; k <= n ; k++) {
❻     st[k] = str.charAt(k);
    }
❼   for (int k = n; k >= 0; k--) {
❽     System.out.printf("%c", st[k]);
    }
  }
}
```

❷ 문자열 변수 str의 크기인 8을 정수형 변수 n의 초기값으로 할당한다.
※ length() 메소드는 문자열 변수에 저장된 문자열의 길이를 반환한다.

❻ 문자열 변수 str에서 k 번째에 있는 문자를 st[k]에 저장한다. 결과적으로, 문자열 변수 str의 값을 st[0]~st[7] 순으로 앞에서부터 차례로 한 글자씩 저장한다.
※ charAt() 메소드는 문자열에서 지정된 위치의 문자를 읽어온다.

결과 gnoganiS

[문제 36]

i=7, hap=9입니다.

```
public class Problem {
  public static void main(String[] args){
    int i = 1, hap = 0;
    while(i <= 6) {
❶     hap += i;
❷     i += 2;
    }
❸   System.out.printf("i=%d, hap=%d입니다.\n", i, hap);
  }
}
```

i가 6보다 작거나 같은 동안 ❶~❷번을 반복 수행한다. i가 6보다 커지면 반복문을 벗어나 제어가 ❸번으로 이동한다.

결과 i=7, hap=9입니다.

디버깅

i	hap
1	0
3	1
5	4
7	9

[문제 37]

a=10, b=20, c=–10

```
public class Problem {
   public static void main(String[] args){
❶     int a, b, c;
❷     a = 10;
❸     b = 20;
❹❿    c = prnt(a, b);
⓫     System.out.printf("a=%d, b=%d, c=%d\n", a, b, c);
   }

❺ static int prnt(int x, int y)
   {
❻     int z;
❼     if (x == y) z = x + y;
❽     else z = x – y;
❾     return(z);
   }
}
```

정수형 변수 a, b를 인수로 하여 prnt() 함수를 호출한다. ❺번으로 이동한다.

결과 a=10, b=20, c=–10

a와 b는 원래의 값 10과 20을 그대로 출력하고 c는 리턴값 –10을 받았으므로 –10을 출력한다.

실행 클래스 안에 함수를 정의할 때는 static을 붙여야 한다.

- int : 함수의 리턴값이 정수형 변수라는 의미이다.
- prnt : 함수의 이름이다. 사용자가 임의로 적으면 된다.
- (int x, int y) : 메소드의 인수로 정수형 x는 a의 값 10을 받고, 정수형 y는 b의 값 20을 받는다.

[문제 38]

9, 5

```
public class Problem {
  public static void main(String[] args){
❶   int[][] a = {  {1, 1, 0, 1, 0},
                   {1, 0, 1, 0},
                };
    int tot = 0;
    int totSu = 0;
❷   for (int i[] : a)
    {
❸     for (int j : i) tot += j;
❹     totSu = totSu + i.length;
    }
❺   System.out.printf("%d, %d\n", totSu, tot);
  }
}
```

다음과 같은 행과 열의 크기를 갖는 정수형 배열 a가 선언되고 다음과 같이 초기화된다.

배열 a[][]

	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
	1	1	0	1	0
	1	0	1	0	
	a[1][0]	a[1][1]	a[1][2]	a[1][3]	

a 배열의 행 수만큼 ❸~❹번을 반복 수행한다.
- int i[] : a 배열의 한 개의 행이 할당될 변수를 1차원 배열로 선언한다.
- a : 배열의 이름을 적어준다. a 배열이 2행이므로 각 행을 일차원 배열 i[]에 할당하면서 ❸~❹번을 2회 수행한다.

i 배열의 요소 수만큼 'tot += j'를 반복 수행한다.
- int j : i 배열의 각 요소가 할당될 변수를 선언한다.
- i : 배열의 이름을 적어준다. i 배열이 5개 혹은 4개의 요소를 가지므로 각 요소를 i에 할당하면서 ❹번을 5번 혹은 4번 수행한다.
- tot += j : 'tot = tot + j'와 동일하다. j를 tot에 누적한다.

i 배열의 크기, 즉 i 배열의 요소 수가 totSu에 누적된다.

결과 9, 5

디버깅

tot	totSu	j	배열 i	배열 a
0	0			
1	5	1		
2		1		
2		0	1 1 0 1 0	
3		1		1 1 0 1 0
3		0		1 0 1 0
4	9	1		
4		0		
5		1	1 0 1 0	
5		0		

[문제 39]

a=40, b=–10, c=50

```
❶ class IntClass{
❷   int a;
❸   int b;
❹   int c;
  }

public class Problem {
  public static void main(String[] args){
❺   IntClass myVar = new IntClass();
❻   myVar.a = 10;
❼   myVar.b = 20;
❽   prnt(myVar);
⓰   System.out.printf("a=%d, b=%d, c=%d\n", myVar.a, myVar.b, myVar.c);
  }

❾ static void prnt(IntClass myVar)
  {
❿   myVar.a += 30;
⓫   myVar.b -= 30;
⓬   if (myVar.a <= myVar.b)
⓭     myVar.c = myVar.a + myVar.b;
⓮   else
⓯     myVar.c = myVar.a - myVar.b;
  }
}
```

IntClass 클래스를 정의한다. class 외부에 선언되었으므로 static을 붙이지 않는다.

IntClass 객체 변수 myVar을 선언한다.

객체 변수 myVar을 인수로 하여 prnt() 함수를 호출한다. ❾번으로 이동한다.

결과 : a=40, b=–10, c=50
myVar.a는 ❿번을 수행한 후의 값인 40을,
myVar.b는 ⓫번을 수행한 후의 값인 –10을,
myVar.c는 ⓬번 조건을 만족하지 못해 ⓯번을
수행한 후의 값인 50을 출력한다.

실행 클래스 안에 함수를 정의할 때는 static을 붙여야 한다.
- void : 리턴값이 없으므로 void를 붙인다.
- prnt : 함수의 이름이다. 사용자가 임의로 적으면 된다.
- (IntClass myVar) : 메소드의 인수로 IntClass 클래스의 객체 변수 myVar을 받는다. ❽에서 호출할 때 사용한 개체 변수명과 동일해도 되고 그렇지 않아도 된다.

디버깅

myVar.a	myVar.b	myVar.c
10	20	50
40	–10	

[문제 40]

합은 6입니다.

```
public class Problem {
   public static void main(String[] args){
❶    int i = 0, hap = 0;
❷    for(i = 1; i <= 5; ++i);
❸    hap += i;
❹    System.out.printf("합은 %d입니다.\n", hap);
   }
}
```

반복 변수 i가 1에서 시작하여 1씩 증가하면서 5보다 작거나 같은 동안 반복한다. ❷번 문장 끝에 세미콜론(;)이 있으므로 별도의 반복 범위 없이 반복 변수만 증가시킨다.

결과 합은 6입니다.

주의할 점은 반복문을 벗어날 때 반복 변수는 'i<=5'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 5보다 작거나 같은 동안에는 반복문을 수행하고, i가 1 증가하여 6이 되었을 때 반복문을 벗어난다.

디버깅

i	반복 횟수	hap
0		0
1	1	6
2	2	
3	3	
4	4	
5	5	
6		

[문제 41]

a=5, b=5, c=15

```
public class Problem {
❶ static int a,b,c;                    정수형 전역 변수 a, b, c를 선언한다. 전역 변수이므로 함수에서 변경되는 값이 전체에 영향을 미친다.
❷ public static void main(String[] args){
❸   a = 0;
    b = 5;
    c = 0;
❹   prnt();                            prnt() 함수를 호출한다. 제어가 ❺번으로 이동한다.
❿   System.out.printf("a=%d, b=%d, c=%d\n", a, b, c);    결과 a=5, b=5, c=15
  }

❺ static void prnt()
  {
❻   while (a < b) {
❼     ++a;                             a의 값을 1 증가시킨다.
❽     c = c + a;                       a의 값을 c에 누적한다.
❾     prnt();
    }
  }
}
```

❾ 여기서부터 자기가 자기를 호출하는 순환 프로그램이 시작된다. 순환 프로그램은 순환하는 만큼 반복하여 실행하면서 변수에 저장된 값을 추적하면 결과를 이해하기 쉽다.

• a b c: 0 5 0 인 상태에서 prnt()를 호출한다.

①회

```
prnt(){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

• 변수의 값이 a b c: 1 5 1 인 상태에서 다시 prnt()를 호출한다.

②회

```
prnt(){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

• 변수의 값이 a b c: 2 5 3 인 상태에서 다시 prnt()를 호출한다.

③회

```
prnt(){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

• 변수의 값이 a b c: 3 5 6 인 상태에서 다시 prnt()를 호출한다.

④회

```
prnt(){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

• 변수의 값이 | a | b | c |
|---|---|---|
| 4 | 5 | 10 |

인 상태에서 다시 prnt()를 호출한다.

⑤회

```
prnt( ){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

• 변수의 값이 | a | b | c |
|---|---|---|
| 5 | 5 | 15 |

인 상태에서 다시 prnt()를 호출한다.

⑥회

```
prnt( ){
  while (a < b) {
    ++a;
    c = c + a;
    prnt(); }
}
```

a의 값이 5가 되었다는 것은 ①회~⑤회에서 비교하는 조건도 모두 a가 5인 상태로 while문의 조건을 비교한다는 것이다.

- 일단 a의 값이 5이므로 ⑥회 prnt() 함수의 while 조건에 어긋나 while문을 빠져나와 함수의 실행을 종료하고, 제어를 ⑤회 prnt() 함수로 옮긴다.
- a의 값은 여전히 5로 유지되므로 ⑤회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ④회 prnt() 함수로 옮긴다.
- 역시 a의 값이 5이므로 ④회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ③회 prnt() 함수로 옮긴다.
- a의 값은 여전히 5로 유지되므로 ③회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ②회 prnt() 함수로 옮긴다.
- a의 값이 5이므로 ②회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ①회 prnt() 함수로 옮긴다.
- a의 값이 5이므로 ①회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 처음 prnt() 함수를 호출한 ❹번으로 돌아가 이어서 ❿번을 수행한다.

[문제 42]

Daejun

```
class CharClass{
   String a[] = { "Seoul","Incheon","Kyonggi","Daejun","Daegu","Pusan" };

}

public class Problem {
   public static void main(String[] args){
      CharClass myVar = new CharClass();
      String str01 = myVar.a[3];
      System.out.printf("%s", str01);
   }
}
```

CharClass 클래스를 정의한다. class 외부에 선언되었으므로 static을 붙이지 않는다.

배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

CharClass 객체 변수 myVar을 선언한다.

myVar.a[3]의 값을 문자열 변수 str01에 저장한다.

결과

```
Daejun
```

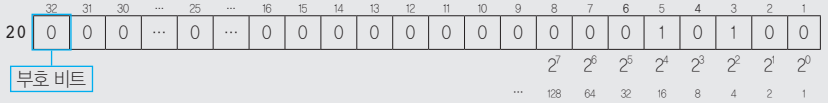
[문제 43]

a=10, b=10

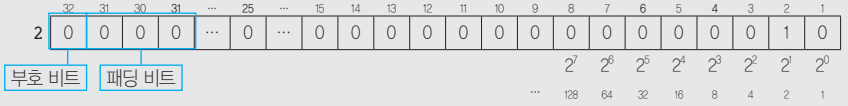
```
public class Problem {
❶ static class IntClass{          IntClass 클래스를 정의한다.
❷   int a, b;
  }

  public static void main(String[] args){
❸   IntClass myVar = new IntClass();          IntClass 객체 변수 myVar을 선언한다.
❹   myVar.a = 10;
❺   myVar.b = 20;
❻   prnt(myVar);                  객체 변수 myVar을 인수로 하여 prnt() 함수를 호출한다. ❼번으로 이동한다.
⓯   System.out.printf("a=%d, b=%d\n", myVar.a, myVar.b);      결과  a=10, b=10
  }

❼ static void prnt(IntClass myVar)
  {
❽   switch (myVar.b >> 3)
    {
❾   case 1: myVar.a = myVar.b; break;
❿   case 2: myVar.b = myVar.a; break;
⓫   case 4: myVar.a -= myVar.b; break;
⓬   case 8: myVar.a += myVar.b; break;
⓭   case 16: myVar.a *= myVar.b; break;
⓮   default: break;
    }
  }     제어가 ⓯번으로 이동된다.
}
```

>>는 오른쪽 시프트 연산자이므로, myVar.b의 값 20을 오른쪽으로 3비트 이동시킨다. int는 4Byte이므로 4Byte 2진수로 변환하여 계산하면 된다.

- 4바이트에 20을 2진수로 표현하면 다음과 같다.



- 부호를 제외한 전체 비트를 오른쪽으로 3비트 이동시킨다. 부호는 맨 왼쪽의 0이다. 양수이므로 빈 자리(패딩 비트)에는 0이 들어오면 된다.



이것을 10진수로 변환하면 2다. 'myVar.b >> 3'의 값이 2이므로 ❿로 이동한다.

❾ 'myVar.b >> 3'의 값이 1인 경우이다. 작업을 수행한 후 switch문을 탈출한다.

❿ 'myVar.b >> 3'의 값이 2인 경우이다. 작업을 수행한 후 switch문을 탈출한다.

⓫ 'myVar.b >> 3'의 값이 4인 경우이다. 작업을 수행한 후 switch문을 탈출한다.

⓬ 'myVar.b >> 3'의 값이 8인 경우이다. 작업을 수행한 후 switch문을 탈출한다.

⓭ 'myVar.b >> 3'의 값이 16인 경우이다. 작업을 수행한 후 switch문을 탈출한다.

⓮ 'myVar.b >> 3'의 값이 1, 2, 4, 8, 16이 아닌 경우이다. switch문을 탈출한다.

[문제 44]

–32768

```
public class Problem {
  public static void main(String[] args){
    short inst = 32767;          short 정수형 변수 inst를 선언하면서 32767을 저장한다. short형의 크기는 2바이트이다.
    inst += 1;                   'inst = inst + 1;'과 동일하다. inst의 값에 1을 누적시킨다.
    System.out.printf("%d", inst);                    결과  –32768
  }
}
```

※ 32768은 short 정수형의 범위를 넘기 때문에 오버플로가 발생한 것이다. –32768이 출력된 이유는 다음과 같다.

JAVA는 C언어와 마찬가지로 부호화 2의 보수법을 사용한다. short 정수형은 길이가 2byte이므로 저장할 수 있는 범위는 32767 ~ –32768인데, 이건 외우지 않아도 된다. 대충 범위가 이정도구나 정도로 알면 된다. 10진수를 2진수로 바꾸고 2진수를 10진수로 바꾸는 방법만 알면 이해할 수 있다.

32767을 2바이트 2진수로 바꾸면 0111 1111 1111 1111이다. 여기에 1을 더하면

```
0111 1111 1111 1111
+                 1
```

1000 0000 0000 0000이다. 2의 보수법에서 맨 왼쪽의 비트는 부호다. 정보처리 필기를 공부하면서 다 배운 내용이다. 즉 맨 왼쪽 비트가 1이라는 것은 그 값이 음수라는 것이다. 음수인 경우 그 값을 알려면 다시 2의 보수로 변환한 다음 10진수로 바꾸고 음의 부호(–)를 붙인다.

2의 보수는 오른쪽에서 시작해서 첫 번째 1이 나올 때까지는 그대로 쓰고 나머지 비트는 반전시키면 된다. 즉 1000 0000 0000 0000은 2의 보수도 그대로 1000 0000 0000 0000이며, 이것은 10진수로 32768이다. 근데 이 값은 원래 음수였던 것을 값을 알기위해 보수를 취한 것이므로 음의 부호를 붙이면 –32768이 되는 것이다.

[문제 45]

i 〈 100

```
public class Problem {
  public static void main(String[] args){
❶   int i = 0, hap = 0;
❷   do{                                  do~while 반복문의 시작점이다. ❸~❹번 문장을 반복 수행한다.
❸     ++i;
❹     hap += i;
❺   } while(i < 100);                    i가 100보다 작은 동안 ❸~❹번 문장을 반복 수행한다.
    System.out.printf("1에서 100까지의 합은 %5d입니다.\n", hap);
  }                          결과  1에서 100까지의 합은 5050입니다.
}                            i가 100이 되었을 때 100을 hap에 누적한 다음 do~while문을 탈출하기 때문에 i는 100으로 끝난다.
```

디버깅

i	hap
0	0
1	1
2	3
3	6
4	10
5	15
⋮	⋮
96	4656
97	4753
98	4851
99	4950
100	5050

[문제 46]

24 12 6 3 3

```
public class Problem {
  public static void main(String[] args){
    int numAry[] = { 0,0,0,0,3 };
    int i, j;
    for (j = 4; j >= 0; --j)
      for (i = 4; i > j; --i)
        numAry[j] += numAry[i];
    for (j = 0; j < 5; ++j)
      System.out.printf("%d ", numAry[j]);
  }
}
```

배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

	첫 번째	두 번째	세 번째	네 번째	다섯 번째
배열 numAry	0	0	0	0	3
	numAry[0]	numAry[1]	numAry[2]	numAry[3]	numAry[4]

numAry[j]의 값을 출력한다.

디버깅

j	i	numAry[i]	numAry[j]	numAry[j] += numAry[i]	배열 numAry	출력
4	4					
3	4	3	0	3		
	3					
2	4	3	0	3		
	3	3	3	6		
	2					
1	4	3	0	3	0 0 0 0 3	
	3	3	3	6	3 3 3 3	
	2	6	6	12	6 6 6	
	1				12 12	
					24	
0	4	3	0	3		
	3	3	3	6		
	2	6	6	12		
	1	12	12	24		
	0					
–1						24 12 6 3 3

[문제 47]

a=5, b=5, c=15

```
#include<stdio.h>
❶ void prnt(int *a, int *b, int *c);

  main() {
❷   int a = 0, b = 5, c = 0;
❸   prnt(&a, &b, &c);
❿   printf("a=%d, b=%d, c=%d\n", a, b, c);
  }

❹ void prnt(x, y, z)
❺ int *x, *y, *z;
  {
❻   while (*x < *y) {
❼     ++*x;
❽     *z = *z + *x;
❾     prnt(x, y, z);
    }
  }
```

❶ 사용할 함수를 선언하는 곳이다.
- void : 리턴값이 없으므로 void를 붙인다.
- (int *a, int *b, int *c) : 함수에서 사용할 인수다. 정수형 포인터 변수 3개를 사용한다는 뜻인데, 호출하는 곳에서 보내준 인수의 순서와 자료형이 일치해야 한다.

❸ 정수형 변수 a, b, c의 주소를 인수로 하여 prnt() 함수를 호출한다. 제어가 ❹번으로 이동한다.

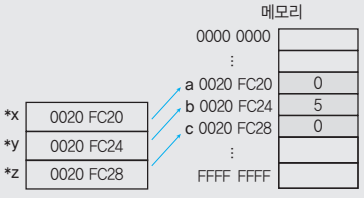
❿ 결과 a=5, b=5, c=15

❺ 인수로 받은 x, y, z가 정수형 변수의 주소를 저장할 수 있는 정수형 포인터 변수라고 선언한다.

❼ x가 가리키는 곳의 값을 1 증가시킨다.

❽ z가 가리키는 곳의 값과 x가 가리키는 곳의 값을 더해서 z가 가리키는 곳에 저장한다.

❹ 함수의 리턴 값이 없으므로 void를 붙인다. ❸번에서 prnt(&a, &b, &c)라고 했으므로 x는 a의 주소를 받고 y는 b의 주소를 받으며, z는 c의 주소를 받는다. 이제 x는 a 변수의 주소를 가리키고, y는 b 변수의 주소를 가리키며, z는 c 변수의 주소를 가리킨다.



❾ 여기서부터 자기가 자기를 호출하는 순환 프로그램이 시작된다. 순환 프로그램은 순환하는 만큼 반복하여 실행하면서 변수에 저장된 값을 추적하면 결과를 알 수 있다.

- 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

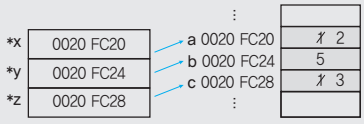
①회

```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```

• 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

②회

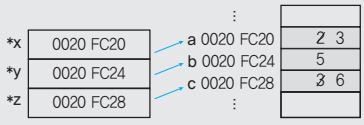
```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```



• 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

③회

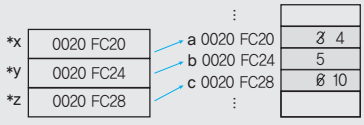
```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```



• 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

④회

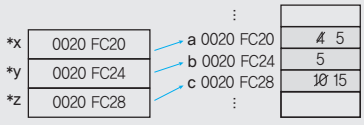
```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```



• 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

⑤회

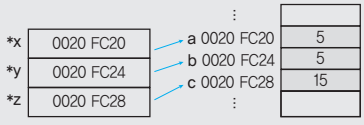
```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```



• 메모리에 저장된 변수의 값이 아래와 같은 상태에서 prnt()를 호출한다.

⑥회

```
prnt( ){
  while (*x < *y) {
    ++*x;
    *z = *z + *x;
    prnt(x, y, z); }
}
```



x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않다는 것은 ①회~⑤회에서 비교하는 조건도 모두 동일한 상태로 while문의 조건을 비교한다는 것이다.

- 일단 x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ⑥회 prnt() 함수의 while 조건에 어긋나 while문을 빠져나와 함수의 실행을 종료하고, 제어를 ⑤회 prnt() 함수로 옮긴다.
- x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ⑤회 prn() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ④회 prnt() 함수로 옮긴다.
- 역시 x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ④회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ③회 prnt() 함수로 옮긴다.
- x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ③회 prn() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ②회 prnt() 함수로 옮긴다.
- x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ②회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 제어를 ①회 prnt() 함수로 옮긴다.
- x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 작지 않으므로 ①회 prnt() 함수에서도 while 조건에 어긋나므로 함수의 실행을 종료하고, 처음 prnt() 함수를 호출한 ❸번으로 돌아간다.

※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

[문제 48]

11, 25

```
public class Problem {
   public static void main(String[] args){
      int i = 1, hap = 0;
      while (i < 10) {                  i가 10보다 작은 동안 ❶~❷번의 문장을 반복 수행한다. i가 10 이상이면 반복문을 벗어나 ❸번으로 이동한다.
❶        hap += i;                      'hap = hap + i;'와 동일하다. i의 값을 hap에 누적시킨다.
❷        i += 2;                        'i = i + 2;'와 동일하다. i의 값을 2씩 누적시킨다.
      }
❸     System.out.printf("%d, %d\n", i, hap);          결과  11, 25
   }
}
```

디버깅

반복 횟수	i	hap
	1	0
1	3	1
2	5	4
3	7	9
4	9	16
5	11	25

[문제 49]

!moT ma I

```
public class Problem {
  public static void main(String[] args){
❶   String str = "I am Tom!";
❷   int n = str.length();
❸   char[] st = new char [n];
❹   char temp;
❺   for (int k = 0; k < n; k++) {
❻     st[k] = str.charAt(k);
    }
❼   n--;
❽   for (int k = 0; k < n; k++) {
❾     temp = st[k];
❿     st[k] = st[n];
⓫     st[n] = temp;
⓬     n--;
    }
⓭   for(char a: st)
⓮     System.out.printf("%s",a);
  }
}
```

❷ 문자열 변수 str에 저장된 문자열의 크기인 9를 정수형 변수 n의 초기값으로 할당한다.
※ length() 메소드는 문자열 변수에 저장된 문자열의 크기를 반환한다.

❻ 문자열 변수 str에서 k번째에 있는 문자를 st[k]에 저장한다. 결과적으로, 문자열 변수 str의 값을 st[0]~st[8] 순으로 앞에서부터 차례로 한 글자씩 저장한다.
※ charAt() 메소드는 문자열에서 지정된 위치의 문자를 읽어온다.

❼ n의 값을 1 감소시킨다. 배열의 위치가 0부터 시작하므로 배열 st는 st[0]~st[8]까지 9개의 문자를 저장하게 된다.

❽ 반복 변수 k가 0에서 시작하여 1씩 증가하면서 n보다 작은 동안 ❾~⓬번을 반복 수행하는데, ⓬번에서 n이 1씩 감소하면서 문자의 끝 위치를 하나씩 앞으로 당긴다. 즉 ❾~⓬번을 4회 반복한다.

❾~⓫ st[k]의 값과 st[n]의 값을 교환한다.

디버깅

k	n	Temp	st[k]	st[n]	st[]								
	9				st[0]	st[1]	st[2]	st[3]	st[4]	st[5]	st[6]	st[7]	st[8]
	8				I		a	m		T	o	m	!
0	8	I	I	!	~~I~~		a	m		T	o	m	~~!~~
			!	I	!								I
1	7	빈칸	빈칸	m	!	~~m~~	a	m		T	o	~~m~~	I
			m	빈칸		m							
2	6	a	a	o	!	m	~~a~~	m		T	~~o~~		I
			o	a			o				a		
3	5	m	m	T	!	m	o	~~m~~		~~T~~	a		I
			T	m				T		m			
4	4				!	m	o	T		m	a		I

[문제 50]

101, 5050입니다.

```
public class Problem {
   public static void main(String[] args){
      int i = 0, hap = 0;
      for(i = 1; i <= 100; ++i)
❶        hap += i;
      System.out.printf("%d, %d입니다.\n", i, hap);
   }
}
```

반복 변수 i가 1에서 시작하여 1씩 증가하면서 100보다 작거나 같은 동안 ❶번을 반복 수행한다.

주의할 점은 반복문을 벗어날 때 반복 변수는 'i<=100'의 결과가 거짓이 되도록 증가한 후 빠져 나간다는 것이다. 여기서는 i가 100보다 작거나 같은 동안에는 반복문을 수행하고, i가 1 증가하여 101이 되었을 때 반복문을 종료한다.

디버깅

i	hap
0	0
1	1
2	3
3	6
4	10
5	15
⋮	⋮
96	4656
97	4753
98	4851
99	4950
100	5050
101	

[문제 51]

15 5

```
#include <stdio.h>
❸ void calc(int *x, int *y)
  {
❹   *y -= *x;
❺   *y = *x > *y ? *x - *y : *y - *x;
  }

  main()
  {
❶   int i = 15, j = 35;
❷   calc(&i, &j);
❻   printf("%d %d\n", i, j);
  }
```

사용할 함수를 선언하는 곳이다. 리턴값이 없으므로 void를 붙인다. 포인터 변수 *x와 *y를 사용하며, ❷번을 참조하면 변수 i와 j의 주소를 가진다는 것을 알 수 있다.

y가 가리키는 곳의 값 35에서 x가 가리키는 곳의 값 15를 뺀 20을 y가 가리키는 곳에 저장한다.

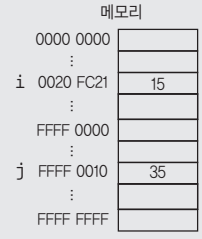
y가 가리키는 곳의 값에 x가 가리키는 곳의 값이 y가 가리키는 곳의 값보다 크면 x가 가리키는 곳의 값에서 y가 가리키는 곳의 값을 빼고, 아니면 y가 가리키는 곳의 값에서 x가 가리키는 곳의 값을 뺀 후 그 값을 저장한다. y가 가리키는 곳의 값은 5가 된다.

정수형 변수 i, j의 주소를 인수로 하여 calc() 함수를 호출한다. ❸번으로 이동한다.

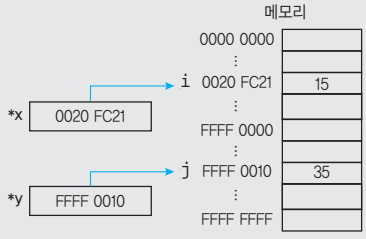
결과 15 5

위 코드의 처리 과정에 따른 메모리의 변화를 그려보면 다음과 같다.

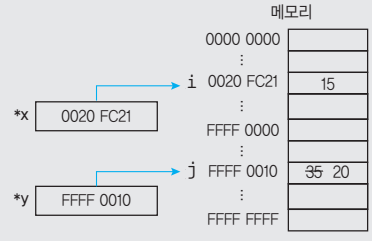
• ❶번 수행(i, j 변수 생성)



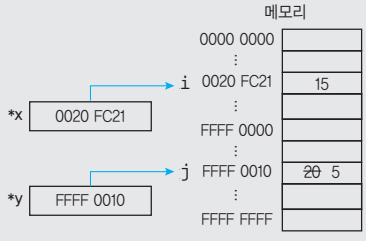
• ❷, ❸번 수행



• ❹번 수행



• ❺번 수행



• ❻번 수행

'15 5' 출력

※ JAVA에서는 포인터 변수를 사용할 수 없습니다.

[문제 52]

30

```
public class Problem{
   public static void main(String[] args) {
❶     int a = 0, sum = 0;
❷     while (a < 10)
      {
❸        a++;
❹        if(a % 2 == 1)
❺           continue;
❻        sum += a;
      }
❼     System.out.println(sum);
   }
}
```

❶ 정수형 변수 a와 sum을 선언한 후 초기값으로 0을 할당한다.

❷ a가 10보다 작은 동안 ❸~❻번 문장을 반복 수행한다.

❸ a = a + 1;과 동일하다. a의 값을 1씩 증가시킨다.

❹ a를 2로 나눈 나머지가 1이면 ❺번을 수행한다.

❺ 제어가 while문의 시작점으로 이동하여 ❸번을 다시 수행한다.

❻ sum = sum + a;와 동일하다. sum에 a를 누적한다.

❼ sum의 값을 출력한 후 커서를 다음 줄의 처음으로 옮긴다.

디버깅

a	sum	a%2	출력
0	0		30
1		1	
2	2	0	
3		1	
4	6	0	
5		1	
6	12	0	
7		1	
8	20	0	
9		1	
10	30	0	

[문제 53]

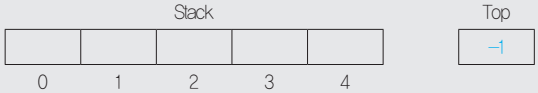
120제거

140제거

```
❶ public class Problem {
❷           static int Stack[] = new int[5];
❸           static int Top = -1;

❹           public static void main(String[] args){
❺              push(100);
❼              push(110);
❾              push(120);
⓫              pop();
⓭              push(130);
⓯              push(140);
⓱              pop();

               }
❻ ❽ ❿ ⓮ ⓰ static void push(int i) {
               ⓐ Top++;
               ⓑ if (Top >= 5)
               ⓒ    System.out.printf("overflow");
               ⓓ else
               ⓔ    Stack[Top] = i;
               }

⓬ ⓲          static void pop() {
               ⓐ if (Top < 0)
               ⓑ    System.out.printf("underflow");
               ⓒ else
               ⓓ    System.out.printf("%d제거\n", Stack[Top--]);
               }
}
```

❷ 5개의 요소를 갖는 정수형 배열 `Stack`를 선언한다.

❸ 전역 변수 `Top`를 선언하면서 초기값으로 -1을 할당한다. 전역 변수이기 때문에 `Problem` 클래스 안에서는 어디서든 사용할 수 있으며 저장된 값이 유지된다.



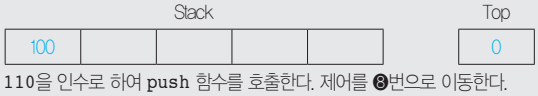
❺ `100`을 인수로 하여 `push` 함수를 호출한다. 제어를 ❻번으로 이동한다.

❻ `push` 함수의 시작점이다. ❺번에서 보낸 `100`을 `i`가 받는다.

ⓐ `Top`의 값을 1 증가 시킨다.

ⓑ `Top`이 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ `Stack[0]`에 `100`을 저장한다.



❼ `110`을 인수로 하여 `push` 함수를 호출한다. 제어를 ❽번으로 이동한다.

❽ ❼번에서 보낸 `110`을 `i`가 받는다.

ⓐ `Top`의 값을 1 증가 시킨다.

ⓑ `Top`이 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[1]에 110을 저장한다.

Stack

100	110			

Top

1

출력결과

❾ 120을 인수로 하여 push 함수를 호출한다. 제어를 ❿번으로 이동한다.

❿ ❾번에서 보낸 120을 i가 받는다.

ⓐ Top의 값을 1 증가 시킨다.

ⓑ Top이 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[2]에 120을 저장한다.

Stack

100	110	120		

Top

2

출력결과

⓫ pop 함수를 호출한다. 제어를 ⓬번으로 이동한다.

ⓐ Top이 0 보다 작지 않으므로 ⓓ번으로 이동한다.

ⓓ 출력 : "120제거"를 출력하고 다음 줄로 커서를 옮긴다.

- Top--는 후치 연산자이기 때문에 Stack[2]의 값을 출력한 다음 1 감소한다.
- 출력을 했다고 없어지는 것이 아니고 Top의 위치가 1이 되었기 때문에 다음에 push를 하면 Stack[2]에 값이 저장되므로 자연스럽게 없어지게 된다. ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

100	110	120		

Top

1

출력결과

120 제거

⓭ 130을 인수로 하여 push 함수를 호출한다. 제어를 ⓮번으로 이동한다.

⓮ ⓭번에서 보낸 130을 i가 받는다.

ⓐ Top의 값을 1 증가 시킨다.

ⓑ Top이 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[2]에 130을 저장한다.

Stack

100	110	130		

Top

2

출력결과

120 제거

⓯ 140을 인수로 하여 push 함수를 호출한다. 제어를 ⓰번으로 이동한다.

⓰ ⓯번에서 보낸 140을 i가 받는다.

ⓐ Top의 값을 1 증가 시킨다.

ⓑ Top이 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[3]에 140을 저장한다.

Stack

100	110	130	140	

Top

3

출력결과

120 제거

⓱ pop 함수를 호출한다. 제어를 ⓭번으로 이동한다.

ⓐ Top이 0 보다 작지 않으므로 ⓓ번으로 이동한다.

ⓓ 출력 : "140제거" 를 출력하고 다음 줄로 커서를 옮긴다.

- Top--는 후치 연산자이기 때문에 Stack[3]의 값을 출력한 다음 1 감소한다.
- 출력을 했다고 없어지는 것이 아니고 Top의 위치가 2가 되었기 때문에 다음에 push를 하면 Stack[3]에 값이 저장되므로 자연스럽게 없어지게 된다. ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

100	110	130	140	

Top

2

출력결과

120 제거

140 제거

[문제 54]

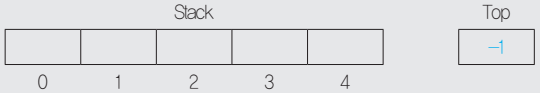
① pop(); ② push('G');

```
❶ #include <stdio.h>
❷ #define MAX 5
❸ char Stack[MAX];
❹ int Top = -1;

❼❾⓫⓭⓯㉑㉓ push(int i) {
              ⓐ Top++;
              ⓑ if (Top >= MAX)
              ⓒ     printf("overflow\n");
              ⓓ else
              ⓔ     Stack[Top] = i;
            }

     ⓱⓳㉕㉗ pop() {
              ⓐ if (Top < 0)
              ⓑ     printf("overflow\n");
              ⓒ else
              ⓓ     printf("%c제거\n", Stack[Top--]);
            }

❺ main() {
❻     push('A');
❽     push('B');
❿     push('C');
⓬     push('D');
⓮     push('F');
⓰     pop();
⓲     pop();
⓴     push('G');
㉒     push('H');
㉔     pop();
㉖     pop();
}
```

❷ 숫자 5를 MAX로 정의한다. 프로그램 안에서 MAX는 숫자 5와 동일하게 쓰인다.

❸ MAX개의 요소, 즉 5개의 요소를 갖는 정수형 배열 Stack을 선언한다.

❹ 전역 변수 Top를 선언하면서 초기값으로 -1을 할당한다. 전역 변수이기 때문에 이 프로그램 안에서는 어디서든 사용할 수 있으며, 변수의 값이 유지된다.



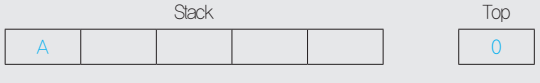
❻ 'A'를 인수로 하여 push 함수를 호출한다. 제어를 ❼번으로 이동한다.

❼ push 함수의 시작점이다. ❻번에서 보낸 'A'를 i가 받는다.

ⓐ Top의 값을 1 증가시킨다.

ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[0]에 'A'를 저장한다.



❽ 'B'를 인수로 하여 push 함수를 호출한다. 제어를 ❾번으로 이동한다.
❾ ❽번에서 보낸 'B'를 i가 받는다.
ⓐ Top의 값을 1 증가시킨다.
ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.
ⓔ Stack[1]에 'B'를 저장한다.

Stack

A	B			

Top: 1

출력결과

❿ 'C'를 인수로 하여 push 함수를 호출한다. 제어를 ⓫번으로 이동한다.
⓫ ❿번에서 보낸 'C'를 i가 받는다.
ⓐ Top의 값을 1 증가시킨다.
ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.
ⓔ Stack[2]에 'C'를 저장한다.

Stack

A	B	C		

Top: 2

출력결과

⓬ 'D'를 인수로 하여 push 함수를 호출한다. 제어를 ⓭번으로 이동한다.
⓭ ⓬번에서 보낸 'D'를 i가 받는다.
ⓐ Top의 값을 1 증가시킨다.
ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.
ⓔ Stack[3]에 'D'를 저장한다.

Stack

A	B	C	D	

Top: 3

출력결과

⓮ 'F'를 인수로 하여 push 함수를 호출한다. 제어를 ⓯번으로 이동한다.
⓯ ⓮번에서 보낸 'F'를 i가 받는다.
ⓐ Top의 값을 1 증가시킨다.
ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.
ⓔ Stack[4]에 'F'를 저장한다.

Stack

A	B	C	D	F

Top: 4

출력결과

⓰ pop 함수를 호출한다. 제어를 ⓱번으로 이동한다.
ⓐ Top이 0보다 작지 않으므로 ⓓ번으로 이동한다.
ⓓ 출력 : "F제거"를 출력하고 다음 줄로 커서를 옮긴다.
- Top는 후치 연산자이기 때문에 Stack[4]의 값을 출력한 다음 1 감소한다.
- 출력을 했다고 없어지는 것이 아니고 Top의 위치가 3이 되었기 때문에 다음에 push를 하면 Stack[4]에 값이 저장되므로 자연스럽게 없어지게 된다.
- ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

A	B	C	D	F

Top: 3

출력결과: F제거

⓲ pop 함수를 호출한다. 제어를 ⓳번으로 이동한다.
ⓐ Top이 0보다 작지 않으므로 ⓓ번으로 이동한다.
ⓓ 출력 : "D제거"를 출력하고 다음 줄로 커서를 옮긴다.
- Top는 후치 연산자이기 때문에 Stack[3]의 값을 출력한 다음 1 감소한다.
- ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

A	B	C	D	F

Top: 2

출력결과: F제거, D제거

⓴ 'G'를 인수로 하여 push 함수를 호출한다. 제어를 ㉑번으로 이동한다.
㉑ ⓴번에서 보낸 'G'를 i가 받는다.
ⓐ Top의 값을 1 증가시킨다.
ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.
ⓔ Stack[3]에 'G'를 저장한다.

Stack

A	B	C	G	F

Top: 3

출력결과: F제거, D제거

㉒ 'H'를 인수로 하여 push 함수를 호출한다. 제어를 ㉓번으로 이동한다.

㉓ ㉒번에서 보낸 'H'를 i가 받는다.

ⓐ Top의 값을 1 증가시킨다.

ⓑ Top이 MAX인 5보다 크거나 같지 않으므로 ⓔ번으로 이동한다.

ⓔ Stack[4]에 'H'를 저장한다.

Stack

A	B	C	G	H

Top: 4

출력결과: F제거, D제거

㉔ pop 함수를 호출한다. 제어를 ㉓번으로 이동한다.

ⓐ Top이 0보다 작지 않으므로 ⓓ번으로 이동한다.

ⓓ 출력 : "H제거"를 출력하고 다음 줄로 커서를 옮긴다.

- Top는 후치 연산자이기 때문에 Stack[4]의 값을 출력한 다음 1 감소한다.
- ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

A	B	C	G	H

Top: 3

출력결과: F제거, D제거, H제거

㉕ pop 함수를 호출한다. 제어를 ㉗번으로 이동한다.

ⓐ Top이 0보다 작지 않으므로 ⓓ번으로 이동한다.

ⓓ 출력 : "G제거"를 출력하고 다음 줄로 커서를 옮긴다.

- Top는 후치 연산자이기 때문에 Stack[3]의 값을 출력한 다음 1 감소한다.
- ⓓ번 실행 후 변수의 변화는 다음과 같다.

Stack

A	B	C	G	H

Top: 2

출력결과: F제거, D제거, H제거, G제거

[문제 55]

a[i]

```
public class Test02{
   public static void main(String[] args) {
❶     int a[] = {10, 30, 50, 70, 90};
❷     int i, max, min;
❸     max = a[0];
❹     min = a[0];
❺     for(i=0; i<5; i++) {
❻        if(a[i] > max)
❼           max = a[i];
❽        if(a[i] < min)
❾           min = a[i];
      }
❿     System.out.printf("%d\n", max);
⓫     System.out.printf("%d\n", min);
   }
}
```

❶ 배열을 선언할 때 사용할 개수를 생략하고 초기값을 지정하면, 초기값으로 지정된 값의 수와 같은 크기의 배열이 선언된다.

a[5]	10	30	50	70	90

❷ 정수형 변수 i, max, min을 선언한다.

❸ 변수 max에 a[0], 즉 10을 저장한다.

❹ 변수 min에 a[0], 즉 10을 저장한다.

❺ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ❻~❾번을 반복하여 수행한다.

❻ a[i]가 max보다 크면 ❼번을 수행한다.

❼ max에 a[i] 값을 저장한다.

❽ a[i]가 min보다 작으면 ❾번을 수행한다.

❾ min에 a[i] 값을 저장한다.

❿ max 값을 정수형으로 출력하고 커서를 다음 줄의 처음으로 옮긴다.

⓫ min 값을 정수형으로 출력하고 커서를 다음 줄의 처음으로 옮긴다.

디버깅

i	max	min	출력
0	10	10	90
1	30		10
2	50		
3	70		
4	90		
5			

Section **010**

[문제]

① i = i + 1 ② J = J + i ③ YES ④ NO

i	J	i 〈 100	출력
0	0	YES	100,5050
1	1	YES	
2	3	YES	
3	6	YES	
4	10	YES	
5	15	YES	
6	21	YES	
7	28	YES	
8	36	YES	
9	45	YES	
10	55	·	
·	·	·	
·	·	·	
·	·	YES	
90	4095	YES	
91	4186	YES	
92	4278	YES	
93	4371	YES	
94	4465	YES	
95	4560	YES	
96	4656	YES	
97	4753	YES	
98	4851	YES	
99	4950	NO	
100	5050		

["잠깐만요"의 문제]

i	J	i 〈 99	출력
–1	0	YES	2500
1	1	YES	
3	4	YES	
5	9	YES	
7	16	YES	
9	25	·	
·	·	·	
·	·	·	
·	·	YES	
91	2116	YES	
93	2209	YES	
95	2304	YES	
97	2401	NO	
99	2500		

Section **011**

[유형 1]

① SW = 0 ② i = i + 1 ③ J = J − i ④ SW = 0

i	J	SW	SW = 0	i 〈 100	출력
0	0	0	YES	YES	−50
1	1	1	NO	YES	
2	−1	0	YES	YES	
3	2	1	NO	YES	
4	−2	0	YES	YES	
5	3	1	NO	YES	
6	−3	0	YES	YES	
7	4	1	NO	YES	
8	−4	0	YES	YES	
9	5	1	NO	YES	
10	−5	0	·	·	
·	·	·	·	·	
·	·	·	·	·	
·	·	·	NO	YES	
90	−45	0	YES	YES	
91	46	1	NO	YES	
92	−46	0	YES	YES	
93	47	1	NO	YES	
94	−47	0	YES	YES	
95	48	1	NO	YES	
96	−48	0	YES	YES	
97	49	1	NO	YES	
98	−49	0	YES	YES	
99	50	1	NO	NO	
100	−50	0			

[유형 2]

① i + 1 ② J + i ③ 99 ④ i + 1 ⑤ J − i

i	J	i : 99	출력
0	0	〈	50
1	1	〈	
2	−1	〈	
3	2	〈	
4	−2	〈	
5	3	·	
6	−3	·	
7	4	·	
8	−4	〈	
9	5	〈	
·	·	〈	
·	·	〈	
·	·	=	
·	·		
90	−45		
91	46		
92	−46		
93	47		
94	−47		
95	48		
96	−48		
97	49		
98	−49		
99	50		

[유형 3]

① J = 1 ② i = i + 1 ③ J = J × i × (–1)

i	J	MOD(i,2)	MOD(i,2) = 0	i 〈 100	출력
0	1	1	NO	YES	9.3326E+157
1	–1	0	YES	YES	
2	–2	1	NO	YES	
3	6	0	YES	YES	
4	24	1	NO	YES	
5	–120	0	YES	YES	
6	–720	1	NO	YES	
7	5040	0	YES	YES	
8	40320	1	NO	YES	
9	–362880	0	YES	YES	
10	–3628800	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
.	.	0	YES	YES	
90	–1.4857E+138	1	NO	YES	
91	1.352E+140	0	YES	YES	
92	1.2438E+142	1	NO	YES	
93	–1.1568E+144	0	YES	YES	
94	–1.0874E+146	1	NO	YES	
95	1.033E+148	0	YES	YES	
96	9.9168E+149	1	NO	YES	
97	–9.6193E+151	0	YES	YES	
98	–9.4269E+153	1	NO	YES	
99	9.3326E+155	0	YES	NO	
100	9.3326E+157				

※ 9.3326E+157은 지수 형식으로 표현된 것으로 표현할 값의 범위가 큰 경우에 사용합니다.

'E+157'이란 실제 값의 소수점 위치가 현재의 소수점 위치에서 우측으로 157번째에 있다는 의미입니다.

즉 실제 값은 9332621544394410…으로 마지막 0뒤로 0이 142개가 더 표시됩니다.

Section 012

① i = i + 1　② YES　③ NO　④ J − (i / (i + 1))　⑤ i 〉= 99

i	J	INT(i/2)	i/2	INT(i/2)=i/2	i 〉= 99	출력
0	0	0	0.5	No	No	−0.688172
1	−0.5	1	1	Yes	No	
2	0.1666667	1	1.5	No	No	
3	−0.5833	2	2	Yes	No	
4	0.2166667	2	2.5	No	No	
5	−0.6166667	3	3	Yes	No	
6	0.2404762	3	3.5	No	No	
7	−0.6345238	4	4	Yes	No	
8	0.2543651	4	4.5	No	No	
9	−0.6456349	5	5	Yes	No	
10	0.263456	.	.	.	.	
.	.	.	.	.	.	
.	.	.	.	.	.	
.	.	45	45	Yes	No	
90	0.3013887	45	45.5	No	No	
91	−0.6877418	46	46	Yes	No	
92	0.3015056	46	46.5	No	No	
93	−0.6878561	47	47	Yes	No	
94	0.3016175	47	47.5	No	No	
95	−0.6879658	48	48	Yes	No	
96	0.3017249	48	48.5	No	No	
97	−0.688071	49	49	Yes	No	
98	0.301828	49	49.5	No	Yes	
99	−0.688172					

Section **013**

[유형 1]

① J = 1 ② K = 1 ③ J = J + i ④ i 〈 19

i	J	K	i 〈 19	출력
0	1	1	Yes	1350
1	2	3	Yes	
2	4	7	Yes	
3	7	14	Yes	
4	11	25	Yes	
5	16	41	Yes	
6	22	63	Yes	
7	29	92	Yes	
8	37	129	Yes	
9	46	175	Yes	
10	56	231	Yes	
11	67	298	Yes	
12	79	377	Yes	
13	92	469	Yes	
14	106	575	Yes	
15	121	696	Yes	
16	137	833	Yes	
17	154	987	Yes	
18	172	1159	No	
19	191	1350		

[유형 2]

① K = –1 ② L = –1 ③ L = L × (–1) ④ i 〈 19

i	J	L	K	i 〈 19	출력
0	1	–1	–1	YES	100
1	2	1	1	YES	
2	4	–1	–3	YES	
3	7	1	4	YES	
4	11	–1	–7	YES	
5	16	1	9	YES	
6	22	–1	–13	YES	
7	29	1	16	YES	
8	37	–1	–21	YES	
9	46	1	25	YES	
10	56	–1	–31	YES	
11	67	1	36	YES	
12	79	–1	–43	YES	
13	92	1	49	YES	
14	106	–1	–57	YES	
15	121	1	64	YES	
16	137	–1	–73	YES	
17	154	1	81	YES	
18	172	–1	–91	NO	
19	191	1	100		

Section 014

① 1 ② i + 1 ③ i ④ J ⑤ 10

i	J	K	i < 10	출력
1	1	1	Yes	4037913
2	2	3	Yes	
3	6	9	Yes	
4	24	33	Yes	
5	120	153	Yes	
6	720	873	Yes	
7	5040	5913	Yes	
8	40320	46233	Yes	
9	362880	409113	NO	
10	3628800	4037913		

Section 015

① HAP = 2 ② CNT = 2 ③ A = B ④ B = C

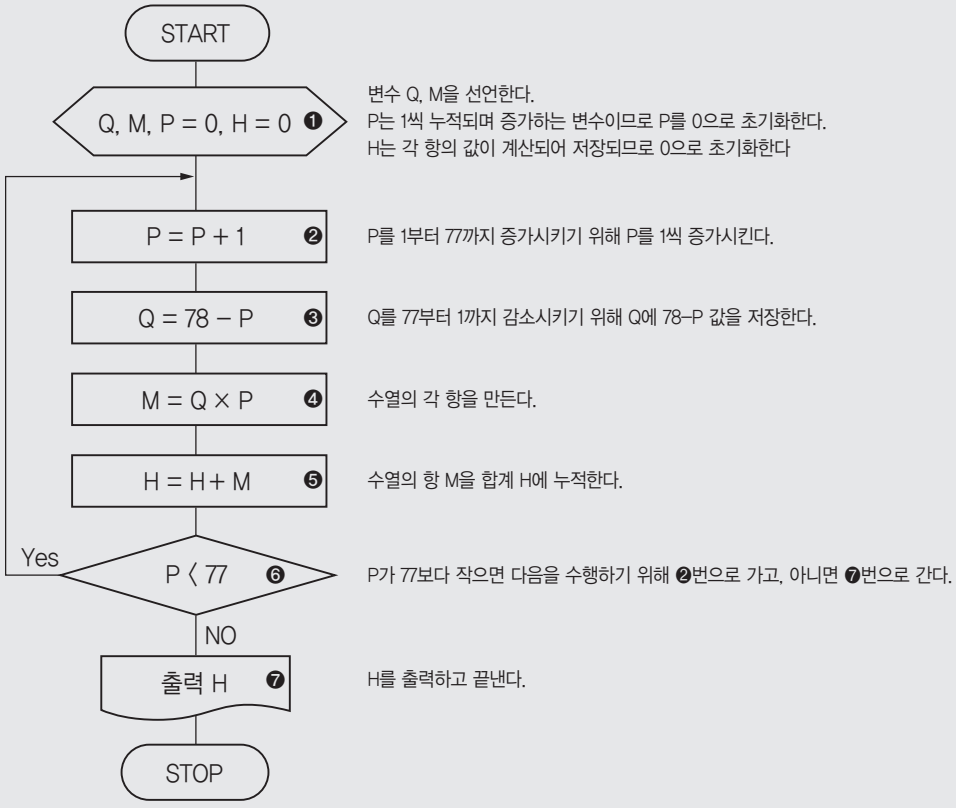
A	B	C	HAP	CNT	CNT 〈 20	출력
1	1	2	2	2	YES	17710
1	2	3	4	3	YES	
2	3	5	7	4	YES	
3	5	8	12	5	YES	
5	8	13	20	6	YES	
8	13	21	33	7	YES	
13	21	34	54	8	YES	
21	34	55	88	9	YES	
34	55	89	143	10	YES	
55	89	144	232	11	YES	
89	144	233	376	12	YES	
144	233	377	609	13	YES	
233	377	610	986	14	YES	
377	610	987	1596	15	YES	
610	987	1597	2583	16	YES	
987	1597	2584	4180	17	YES	
1597	2584	4181	6764	18	YES	
2584	4181	6765	10945	19	NO	
			17710	20		

[문제 1] Section 011

① P ② 78 – P ③ M ④ H ⑤ P 〈 77

변수설명

- P : 1씩 증가되는 숫자가 저장될 변수, 즉 P는 1, 2, 3, …, 77까지 차례로 변경된다.
- Q : 1씩 감소되는 숫자가 저장될 변수, 즉 Q는 77, 76, 75, …, 1까지 차례로 변경된다.
- M : 각 항의 값이 저장될 변수, 즉 (77×1), (76×2), … (1×77)의 값이 차례로 저장된다.
- H : 각 항의 값이 계산되어 저장될 변수, 즉 (77×1) + (76×2) + … + (1×77)까지의 계산 값이 저장된다.

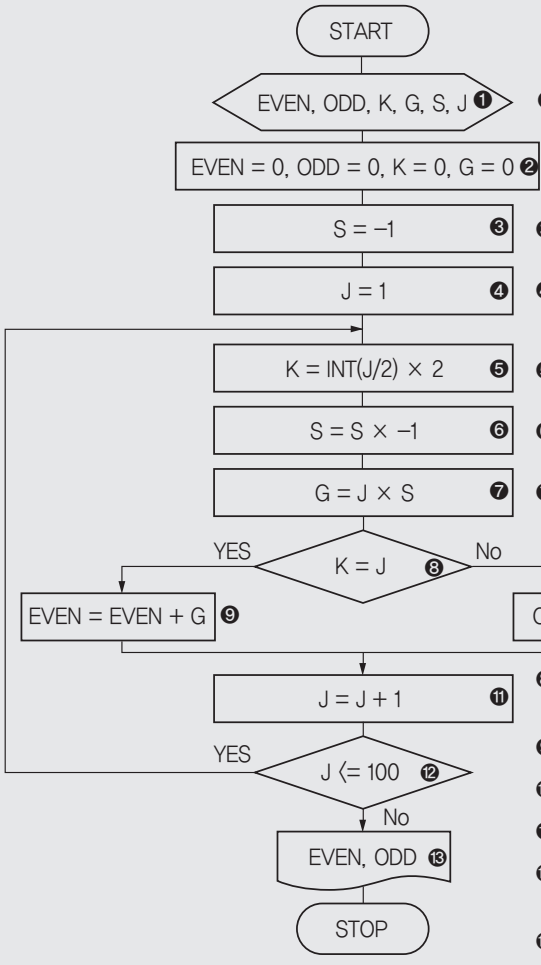


[문제 2] Section 011

① K ② –1 ③ J ④ EVEN = EVEN + G ⑤ ODD = ODD + G ⑥ J = J + 1

변수설명

- ODD : 양수의 합이 저장될 변수(홀수)
- EVEN : 음수의 합이 저장될 변수(짝수)
- J : 1씩 증가되는 숫자가 저장될 변수, 즉 J는 1, 2, 3, …, 100까지 차례로 변경된다.
- K : 홀수인지 짝수인지를 판별할 값이 저장될 변수
- G : J와 S의 곱이 저장될 변수, 즉 G는 1, –2, 3, –4, …, +99, –100까지 차례로 변경된다.
- S : –1과 1을 반복하는 스위치 변수



❶ 사용할 변수를 선언한다.

❷ 음수와 양수가 누적될 변수인 EVEN, ODD를 0으로 초기화한다. K, G는 값을 누적하는 변수가 아니므로 0으로 초기화할 필요가 없는 변수인데, 아마도 혼란을 주기위해 초기화한 것 같다.

❸ 첫 번째 항은 양수 항인데 ❻번에서 –1이 곱해지므로 –1로 초기화한다.

❹ 첫 번째 항을 1로 만든다.

❺ 홀짝을 판별할 값을 계산한다. 어떤 정수를 2로 나눈 몫에 다시 2를 곱할 경우 짝수면 원래의 수와 같고 홀수면 원래의 수 보다 1작다.

❻ 양수와 음수를 결정하기 위한 처리를 한다. –1에 –1을 곱하면 +1이 되고, 1에 –1을 곱하면 –1이 된다.

❼ J와 S의 곱을 저장한다. 짝수인 경우 음수가 되고 홀수인 경우 양수가 된다.

❽ 홀짝을 판별한다. K와 J가 같으면 짝수, 즉 음수이므로 음수 값을 누적하는 ❾번으로 가고, 그렇지 않으면 홀수이므로 양수 값을 누적하는 ❿번으로간다.

❾ 음수의 합계를 누적한다.

❿ 양수의 합계를 누적한다.

⓫ J를 1부터 100까지 증가시키기 위해 1씩 증가시킨다.

⓬ J가 100보다 작거나 같으면 다음을 수행하기 위해 ❺번으로 가고, 아니면 ⓭번으로 간다.

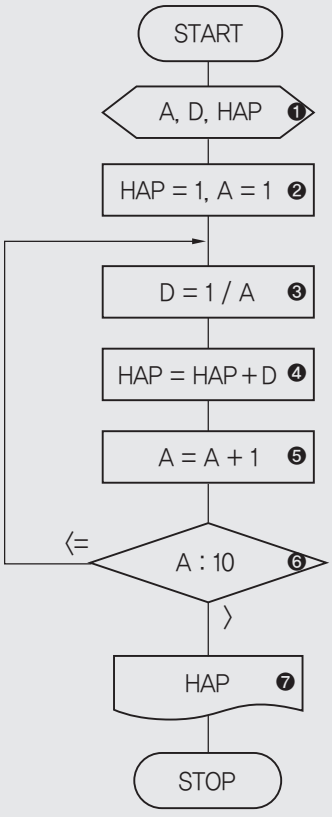
⓭ EVEN과 ODD를 출력하고 끝낸다.

[문제 3] Section 012

① A ② HAP + D ③ A + 1 ④ <= ⑤ >

변수설명

- A : 1씩 증가되는 숫자인 분모 값이 저장될 변수, 즉 A는 1, 2, 3, …, 10까지 차례로 변경된다.
- D : 1/A 형태의 분수가 저장될 변수, 즉 1/1, 1/2, 1/3, 1/4 … 1/10의 값이 저장된다.
- HAP : 1/A의 값이 계산되어 저장될 변수, 즉 1 + 1/1 + 1/2 + 1/3 + 1/4 … +1/10의 계산 값이 저장된다.



변수 A, D, HAP을 선언한다.

처음 더해지는 값 1을 가지고 시작하기 위해 HAP을 1로 초기화한다. 처음 분모 값이 1이므로 A를 1로 초기화한다.

각 항의 분수(1/1, 1/2, …)를 만들어 D에 저장한다.

합계 HAP에 D를 누적한다.

분모를 증가시킨다.

A가 10보다 작거나 같으면 다음을 수행하기 위해 ❸번으로 가고, 아니면 ❼번으로 간다.

HAP을 출력하고 끝낸다.

[문제 4] Section 011

① k == j　② j <= 100

변수설명

- odd : 양수의 합이 저장될 변수(홀수)
- even : 음수의 합이 저장될 변수(짝수)
- **j** : 1씩 증가되는 숫자가 저장될 변수, 즉 j는 1, 2, 3, …, 100까지 차례로 변경된다.
- k : 홀수인지 짝수인지를 판별할 값이 저장될 변수
- g : j와 s의 곱이 저장될 변수, 즉 g는 1, –2, 3, –4, …, +99, –100까지 차례로 변경된다.
- s : –1과 1을 반복하는 스위치 변수

```
#include <stdio.h>

main()
{
❶ int even, odd, k, g, s, j;          정수형 변수 even, odd, k, g, s, j를 선언한다.

❷ even = 0, odd = 0, k = 0, g = 0;    음수와 양수가 누적될 변수인 even, odd를 0으로 초기화한다.
                                      k, g는 값을 누적하는 변수가 아니므로 0으로 초기화할 필요가 없는 변수인데, 아마도 혼란을 주기위해
                                      초기화한 것 같다.
❸ s = -1, j = 1;                      첫 번째 항은 양수 항인데 ❺번에서 –1이 곱해지므로 s를 –1로 초기화한다.
                                      첫 번째 항을 1로 만들기 위해 j를 1로 초기화한다.
  do                                  do~while 반복문의 시작점으로 ❹~❿번을 반복 수행한다.
  {
❹   k = j / 2 * 2;                    홀짝을 판별할 값을 계산한다. 어떤 정수를 2로 나눈 몫에 다시 2를 곱할 경우 짝수면 원래의 수와 같고 홀수면
                                      원래의 수 보다 1작다.
❺   s *= -1;                          양수와 음수를 결정하기 위한 처리를 한다. –1에 –1을 곱하면 +1이 되고, 1에 –1을 곱하면 –1이 된다.
❻   g = j * s;                        j와 s의 곱을 저장한다. 짝수인 경우 음수가 되고 홀수인 경우 양수가 된다.
❼   if (k == j)                       홀짝을 판별한다. k와 j가 같으면 짝수, 즉 음수이므로 음수 값을 누적하는 ❽번으로 가고, 그렇지 않으면 홀수
                                      이므로 양수 값을 누적하는 ❾번으로 간다.
❽     even += g;                      음수의 합계를 누적한다.
    else
❾     odd += g;                       양수의 합계를 누적한다.
❿   j++;                              j를 1부터 100까지 증가시키기 위해 1씩 증가시킨다.
⓫ } while (j <= 100);                 j가 100보다 작거나 같은 동안 ❹~❿번을 반복 수행한다.
⓬ printf("%d %d", even, odd);         even과 odd를 출력하고 끝낸다.
}
```

[문제 5] Section 011

① p++ 또는 p = p + 1　② p < 77

변수설명

- p : 1씩 증가되는 숫자가 저장될 변수, 즉 p는 1, 2, 3, …, 77까지 차례로 변경된다.
- q : 1씩 감소되는 숫자가 저장될 변수, 즉 q는 77, 76, 75, …, 1까지 차례로 변경된다.
- m : 각 항의 값이 저장될 변수, 즉 (77×1), (76×2), … (1×77)의 값이 차례로 저장된다.
- h : 각 항의 값이 계산되어 저장될 변수, 즉 (77×1) + (76×2) + … + (1×77)까지의 계산 값이 저장된다.

```
#include <stdio.h>

main()
{
❶ int q, m;              변수 q, m를 선언한다.

   int p = 0, h = 0;                  p는 1씩 누적되며 증가하는 변수이므로 p를 0으로 초기화한다.
                                      h는 각 항의 값이 계산되어 저장되므로 0으로 초기화한다.
   do                    do~while 반복문의 시작점으로 ❷~❺번을 반복 수행한다.
   {
❷    p++;                           p를 1부터 77까지 증가시키기 위해 p를 1씩 증가시킨다.
❸    q = 78 - p;                    q를 77부터 1까지 감소시키기 위해 q에 78-P 값을 저장한다.
❹    m = q * p;                     수열의 각 항을 만든다.
❺    h += m;                        수열의 항 m을 합계 h에 누적한다.
❻ } while (p < 77);                 p가 77보다 작은 동안 ❷~❺번 문장을 반복 수행한다.
❼ printf("%d", h);                  h를 출력하고 끝낸다.
}
```

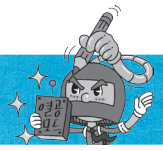
[문제 6] Section 012

① d　② a <= 10

변수설명

- a : 1씩 증가되는 숫자인 분모 값이 저장될 변수, 즉 a는 1, 2, 3, …, 10까지 차례로 변경된다.
- d : 1/a 형태의 분수가 저장될 변수, 즉 1/1, 1/2, 1/3, 1/4 … 1/10의 값이 저장된다.
- hap : 1/a의 값이 계산되어 저장될 변수, 즉 1 + 1/1 + 1/2 + 1/3 + 1/4 … +1/10의 계산 값이 저장된다.

```
#include <stdio.h>

main()
{
❶ float hap, a, d;                    실수형 변수 hap, a, d를 선언한다.

❷ hap = 1, a = 1;                     처음 더해지는 값 1을 가지고 시작하기 위해 hap을 1로 초기화한다. 처음 분모 값이 1이므로 a를 1로 초기화한다.
  do                    do~while 반복문의 시작점으로 ❸~❻번을 반복 수행한다.
  {
❸   d = 1 / a;                        각 항의 분수(1/1, 1/2, …)를 만들어 d에 저장한다.
❹   hap += d;                         합계 hap에 d를 누적한다.
❺   a++;                              분모를 증가시킨다.
❻ } while (a <= 10);                  a가 10보다 작거나 같은 동안 ❸~❻번을 반복 수행한다.
❼ printf("%f", hap);                  hap을 출력하고 끝낸다.
}
```



Section **016**

[유형 1]

① i = A – 1 ② J = 2 ③ 〈= ④ 〉

A	i	J	J : i	MOD(A,J)	MOD(A,J)=0	출력
11	10	2	〈	1	NO	"소수"
		3	〈	2	NO	
		4	〈	3	NO	
		5	〈	1	NO	
		6	〈	5	NO	
		7	〈	4	NO	
		8	〈	3	NO	
		9	〈	2	NO	
		10	=	1	NO	
		11	〉			

[유형 2]

① J = 2 ② J = J + 1 ③ A = J

A	J	MOD(A,J)	MOD(A,J) = 0	A = J	출력
11	2	1	NO	YES	"소수"
	3	2	NO		
	4	3	NO		
	5	1	NO		
	6	5	NO		
	7	4	NO		
	8	3	NO		
	9	2	NO		
	10	1	NO		
	11	0	YES		

[유형 3]

① J = 2 ② NO ③ YES ④ MOD(A, J) = 0

A	J	SQR(A)	MOD(A,J)	출력
17	2	4.123105626	1	"소수"
	3		2	
	4		1	
	5			

Section 017

① J = 2 ② K = J ③ K 〈 A ④ K = K + 1

A	K	J	MOD(K,J)	HAP	출력
7	2	2	0	0	17
	3	2	1	2	
	4	3	0	5	
	5	2	0	10	
	6	2	1	17	
	7	3	2		
		4	1		
		5	0		
		2	0		
		2	1		
		3	1		
		4	3		
		5	2		
		6	1		
		7	0		

Section 018

① A[K-1] = K ② A[i] = 0 ③ M = i ④ M = M + A[i]

i	J	A[i]	M	배열 A	출력
0	0	2	1		5
1	1	3	3		
2	2	0	5		
3	3	5	7		
4	4	0	9		
5	5	7	11		
6		0	2		
7		0	5		
8		0	8	2 3 0 5 0 7 0 0 0 11	
9		11	11		
10			4		
11			9		
			14		
			6		
			13		
			10		
			21		

Section 019

① < ② > ③ BIG - MOK × SMALL ④ NMG = 0

A	B	BIG	SMALL	MOK	NMG	GCM	LCM	출력
15	12	15	12	1	3	3	60	3, 60
		12	3	4	0			

Section **020**

① 〈= ② 〉 ③ D = D + 1 ④ A[D] = C

B	C	MOK	NMG	D	A[D]	출력
28	0	28	0	0		28, 1, 2, 4, 7, 14, 28
	1	14	0	1		
	2	9	1	2		
	3	7	0	3		
	4	5	3	4		
	5	4	4	5		
	6	4	0	6	1 2 4 7 14 28	
	7	3	4			
	8	3	1			
	9	2	8			
	10	2	6			
	11	2	4			
	12	2	2			
	13	2	0			
	14	1	13			
	15	1	12			
	16	1	11			
	17	1	10			
	18	1	9			
	19	1	8			
	20	1	7			
	21	1	6			
	22	1	5			
	23	1	4			
	24	1	3			
	25	1	2			
	26	1	1			
	27	1	0			
	28					
	29					

Section 021

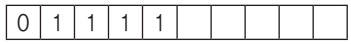
① D = 2 ② D = B ③ NMG = 0 ④ B = MOK

B	D	E	MOK	NMG	C	A[C]	출력
30	2	5	15	0	0	2	2, 3, 5
15	3	3	7	1	1	3	
5	5	2	5	0	2	5	
					3		

Section 022

① BB = B ② A[C] = NMG ③ B = MOK ④ i = C, 1, –1

B	BB	C	MOK	NMG	A[C]	배열 A	출력
30	30	0	15	0	0		30, 11110
15		1	7	1	1		
7		2	3	1	1		
3		3	1	1	1		
1		4	0	1	1		
		5					



Section 023

① <= ② > ③ D = D × B ④ A[E+1] ⑤ C = F

B	C	D	E	F	출력
2	23	1	1	7	1
	7	2	0	7	0
	7	4	1	3	1
	3	8	1	1	1
	1	16	1	0	1
		32			
		16			
		8			
		4			
		2			
		1			

Section 024

① 〉 ② 〈= ③ E = D×2^(5–C)

A	B	C	D	E	출력
1010111011	0	0	1	16	21.84375
	16	1	0	0	
	16	2	1	4	
	20	3	0	0	
	20	4	1	1	
	21	5	1	0.5	
	21.5	6	1	0.25	
	21.75	7	0	0	
	21.75	8	1	0.0625	
	21.8125	9	1	0.03125	
	21.84375	10			
		11			

Section 025

[유형 1]

① J 〈= 10 ② 〉 ③ 〈=

i	A[i]	MAX	출력
1	70	0	95
2	82	70	
3	43	82	
4	90	90	
5	65	95	
6	55		
7	95		
8	45		
9	68		
10	72		

[유형 2]

① i = 1 ② AVG = HAP / 5 ③ HAP = HAP + A[i]

i = 1	A[i]	HAP	MIN	MAX	AVG	출력
1	80	75	75	75	74	25, 100, 370, 74
2	25	155(75+80)	25	80		
3	80	180(75+80+25)		100		
4	70	260(75+80+25+80)				
5	100	330(75+80+25+80+70)				
6	65	430(75+80+25+80+70+100)				
7		495(75+80+25+80+70+100+65)				
		370(495–25–100)				

Section **026**

① i– MOK × 5 ② NMG = 0 ③ HAP = HAP + i

CNT	HAP	i	MOK	NMG	출력
0	0	1	0	1	20, 1050
		2	0	2	
		3	0	3	
		4	0	4	
		5	1	0	
1	5	6	1	1	
		7	1	2	
		8	1	3	
		9	1	4	
		10	2	0	
2	15	11	2	1	
		12	2	2	
		13	2	3	
		14	2	4	
		15	3	0	
.	.	.	.	.	
.	.	.	.	.	
.	.	.	.	.	
18	855	91	18	1	
		92	18	2	
		93	18	3	
		94	18	4	
		95	19	0	
19	950	96	19	1	
		97	19	2	
		98	19	3	
		99	19	4	
		100	20	0	
20	1050				

Section 027

① J = 9　② >=　③ <　④ J = L

J	K	A[K]	L	M	출력
9	1	1	6	1	6
6	2	2	5	2	
5	3	3	4	3	
4	4	4	3	4	
3	5	5	2	5	
2	6	6	1	6	
1	7	8	1	8	
	8	9	2	6	
	9	3	4		
	10	6	1		

Section 028

[유형 1]

① 1　② 1-A[i]　③ i　④ 2　⑤ C

i	C	A	B1	B2	출력
0	1	0	1	2	
1	1	0 1	1 0	0	A
2	1	0 1 1	1 0 0	2 0	0 1 1 0 0
3	0	0 1 1 0	1 0 0 1	0	
4	0	0 1 1 0 0	1 0 0 1 1	1 0 0	B1
5	0			0 1 0 0	1 0 0 1 1
4				0	
3				1 0 1 0 0	B2
2				1	1 0 1 0 0
1					
0					

[유형 2]

① 5, 1, –1 ② 1 ③ A[i] ④ A[i] ⑤ 1–A[J]

i	J	A	B	출력
5	3	0	0	A: 0 1 0 1 0
4	2	1 0	1 0	
	1	0 1 0	1 1 0	
		1 0 1 0	0 1 1 0	B: 1 0 1 1 0
		0 1 0 1 0	1 0 1 1 0	

[유형 3]

① A[i]–C ② ABS(B[i]) ③ 1–B[i]

i	C	A	B	출력
5	1	0	–1	A: 1 0 1 0 0
4	1	0 0	–1 1	
3	1	1 0 0	0 1 1	B: 0 1 1 0 0
2	0	0 1 0 0	0 0 1 1	
1	0	1 0 1 0 0	1 0 0 1 1	
	0			
1			1 0 0 1 1	
2			0 0 0 1 1	
3			0 1 0 1 1	
4			0 1 1 1 1	
5			0 1 1 0 1	

Section 029

① INPUT[2] ② CONT[i] ③ CONT[i+1] ④ INPUT[i+2] ⑤ CONT[i+1]

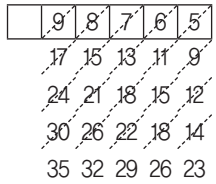
i	INPUT	CONT	출력
1	0	1	
2	0 1	1 0	
3	0 1 1	1 0 0	
4	0 1 1 1	1 0 0 1	
5	0 1 1 1 0		그레이 코드: 1 0 0 1
1			
2			
3			
1			
2			
3			
4			

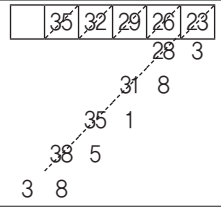
Section **030**

[유형 1]

① B[i] – MOK × 10 ② B[i] = NMG ③ B[1] = 0

i	A[i]	B[i+1]	MOK	NMG	B[i]	B[i–1]	J	배열 B
1	9	9						
2	8	8						
3	7	7						
4	6	6						
5	5	5						
1	8	17						
2	7	15						
3	6	13						
4	5	11						
5	4	9						
1	7	24						
2	6	21						
3	5	18						
4	4	15						
5	3	12						
1	6	30						
2	5	26						
3	4	22						
4	3	18						
5	2	14						
1	5	35						
2	6	32						
3	7	29						
4	8	26						
5	9	23						
1	0							
2	0							
3	0							
4	0							
5	0							
6			2	3	3	28		
5			2	8	8	31		
4			3	1	1	35		
3			3	5	5	38		
2			3	8	8	3	1	
1					3			
2					8			
3					5			
4					1			
5					8			
6					3			





[유형 2]

① Z ② Z–2 ③ A[J]=Z ④ J–1 ⑤ C

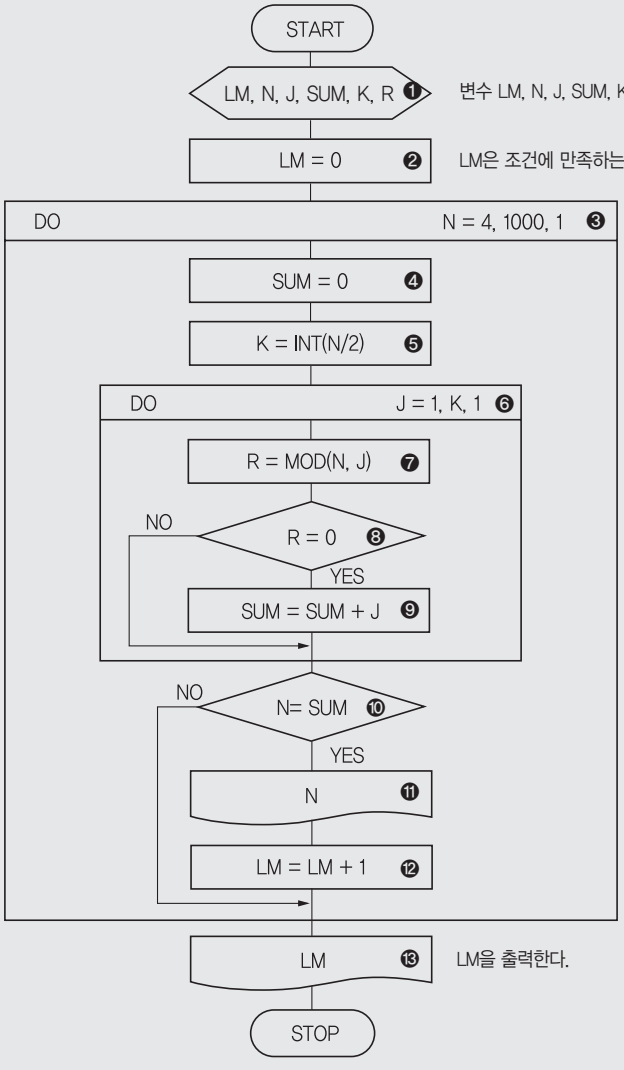
Z	C	J	X[J]	Y[J]	A
0	0	5	0	1	[ ][ ][ ][ ][1]
1	0	4	1	0	[ ][ ][ ][1][1]
1	0	3	1	1	[ ][ ][0][1][1]
2	1	2	0	0	[ ][1][0][1][1]
1	0	1	1	0	[1][1][0][1][1]
1	0				
		0			1
		1			1
		2			0
		3			1
		4			1
		5			

[문제 1] Section 020

① MOD(N, J) ② 0 ③ SUM + J ④ SUM ⑤ LM + 1

변수설명

- LM : 자신을 제외한 약수의 합이 자신과 같은 수의 개수를 구하는 변수
- N, J, R : 계산 처리를 위한 변수
- SUM : 약수들의 합을 구하는 변수
- K : 어떤 수의 모든 약수에서 자신을 제외한 약수 중 최대값



❶ 변수 LM, N, J, SUM, K, R을 선언한다.

❷ LM은 조건에 만족하는 수의 개수가 누적되므로 LM을 0으로 초기화한다.

❸ 4부터 1000까지의 정수를 대상으로 하기 위해 반복 변수 N을 4부터 1000까지 1씩 증가하면서 ❹~⓬번을 반복하여 수행한다.

❹ 약수들의 합을 구하는 변수 SUM을 0으로 초기화한다.

❺ N의 약수 중 최대값을 구하기 위해 N을 2로 나눈 몫을 K에 저장한다.

❻ 안쪽 반복문은 N의 약수를 구한다. J가 1에서 K가 될 때까지 1씩 증가하면서 N의 약수 인지를 계산한다.

❼ N을 J로 나눈 나머지를 R에 저장한다.

❽ R이 0이면 약수이므로 ❾번으로 가고 아니면 ❻번으로간다.

❾ 약수 J를 SUM에 누적시킨다.

❿ N과 SUM이 같으면 조건에 맞는 약수를 한 개 찾았으므로 ⓫번으로 가고 아니면 다음 수를 계산하기 위해 ❸번으로간다.

⓫ N을 출력한다.

⓬ 자신을 제외한 약수의 합이 자신과 같은 수의 개수를 나타내는 LM을 1 증가시킨다.

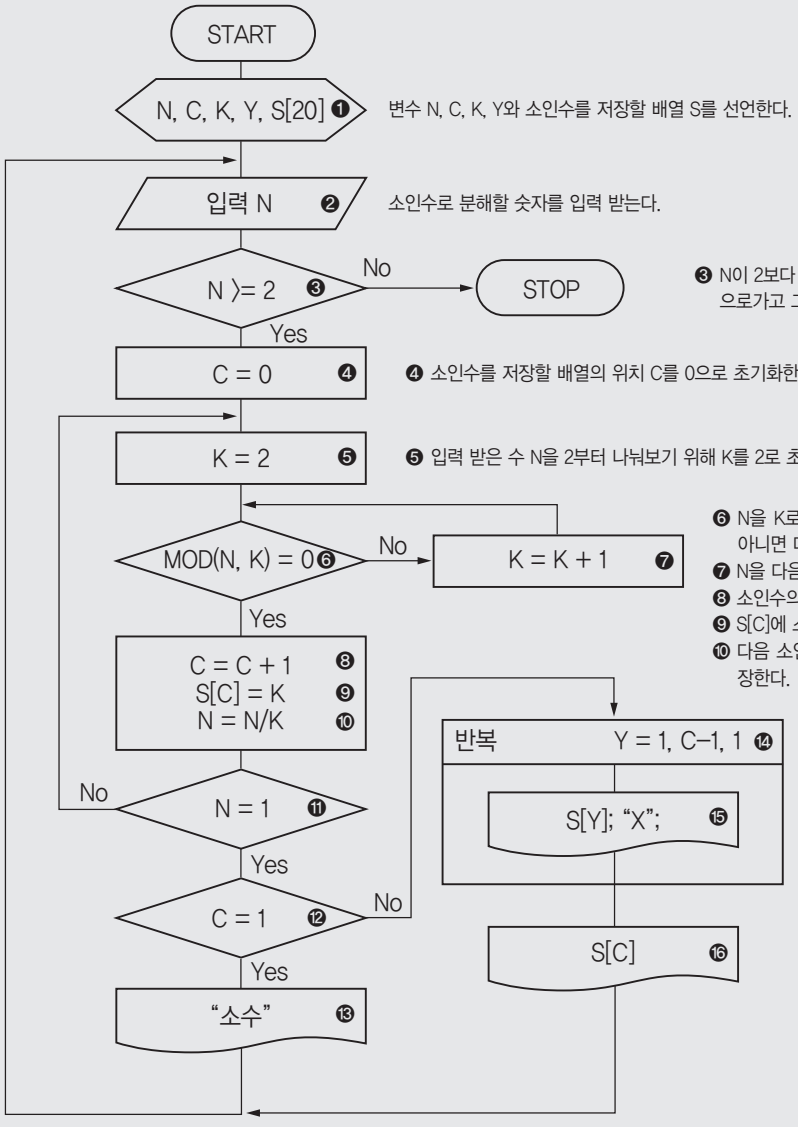
⓭ LM을 출력한다.

[문제 2] Section 021

① 0(Zero)　② K　③ N/K　④ 1　⑤ C

변수설명

- S[20] : 소인수가 저장될 배열
- N : 소인수로 분해하기 위해 입력 받은 숫자가 저장될 변수
- C : 소인수를 저장할 배열 S의 위치를 지정해 주는 변수
- K : 제수가 저장될 변수
- Y : 소인수 분해한 결과를 출력할 때 사용할 임시 변수



❶ 변수 N, C, K, Y와 소인수를 저장할 배열 S를 선언한다.

❷ 소인수로 분해할 숫자를 입력 받는다.

❸ N이 2보다 크거나 같으면 소인수 분해를 수행하기 위해 ❹번으로가고 그렇지 않으면 종료한다.

❹ 소인수를 저장할 배열의 위치 C를 0으로 초기화한다.

❺ 입력 받은 수 N을 2부터 나눠보기 위해 K를 2로 초기화한다.

❻ N을 K로 나눈 나머지가 0이면 소인수이므로 ❽번으로가고, 아니면 다음 수로 나눠보기 위해 ❼번으로 간다.
❼ N을 다음 수로 나눠보기 위해 K를 1 증가시킨다.
❽ 소인수의 개수를 세기 위해 C를 1 증가시킨다.
❾ S[C]에 소인수 K를 저장한다.
❿ 다음 소인수를 구하기 위해 N을 K로 나눈 몫을 다시 N에 저장한다.

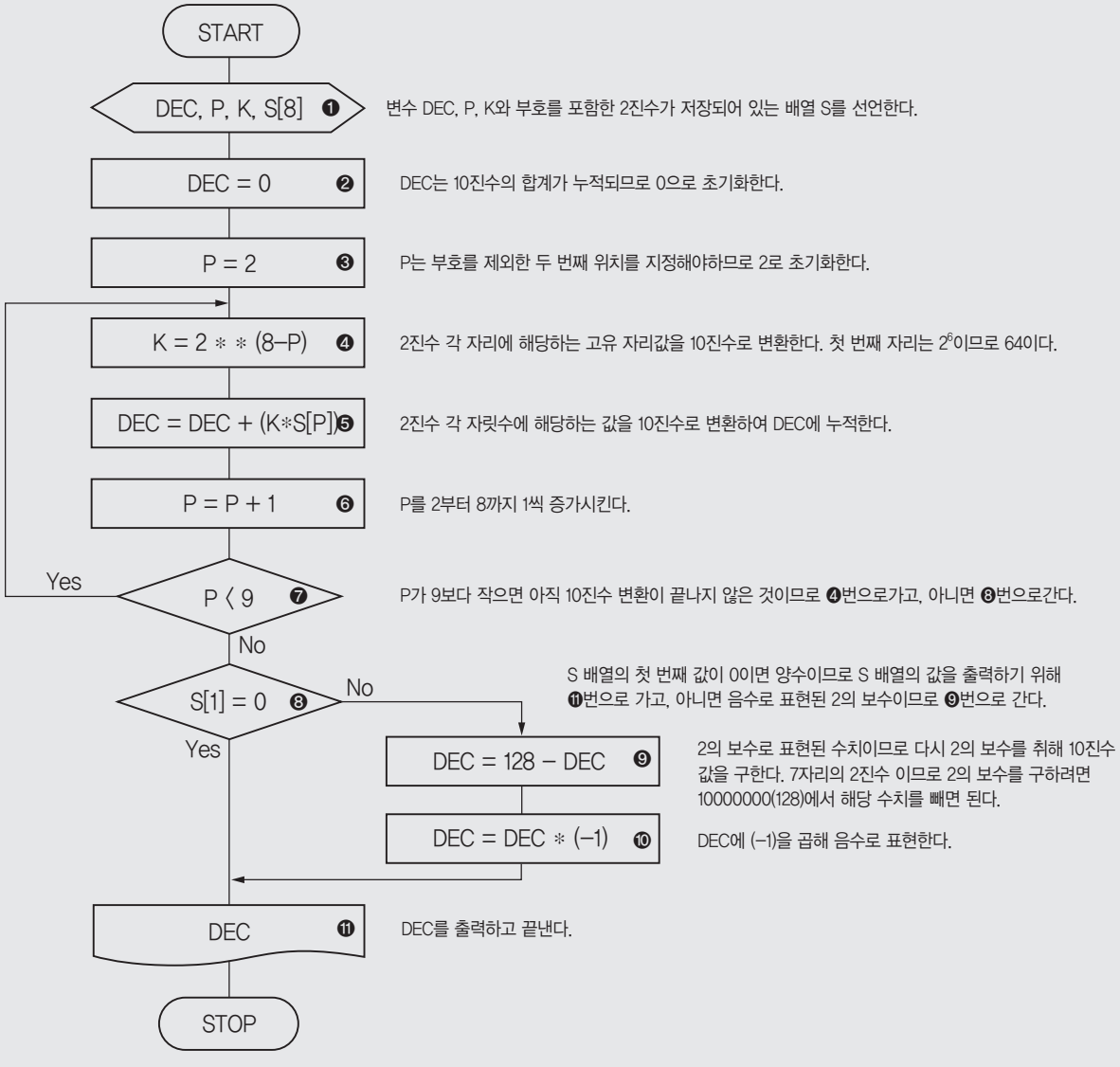
⓫ N이 1이면 소인수 분해 과정을 모두 마친 것으로 ⓬번으로가고, 아니면 ❺번으로간다.
⓬ C가1이면 "소수"이므로 ⓭번으로가고, 아니면 소인수를 출력하기 위해 ⓮번으로간다.
⓭ "소수"를 출력하고 소인수로 분해할 새로운 숫자 를 입력받기 위해 ❷번으로 간다.
⓮ 배열의 위치를 나타내는 반복 변수 Y가 1에서 C-1이 될 때까지 1씩 증가하면서 ⓯번을 반복하여 수행한다.
⓯ S 배열에 저장된 값을 "×" 기호와 함께 출력한다.
⓰ S 배열에 저장된 마지막 값은 "×" 기호 없이 값만 출력한다.

[문제 3] Section 024

① 2 ② (8–P) ③ P = P + 1 ④ S[1] = 0 ⑤ DEC = DEC * (–1)

변수설명

- S[8] : 부호를 포함한 2진수 7자리가 저장되어 있는 배열
- DEC : 2진수 각 자릿수에 대한 10진수의 합계가 저장될 변수
- P : 2진수 각 자리를 지정해 주는 변수
- K : 2진수 각 자리에 대한 10진수 값이 저장될 변수

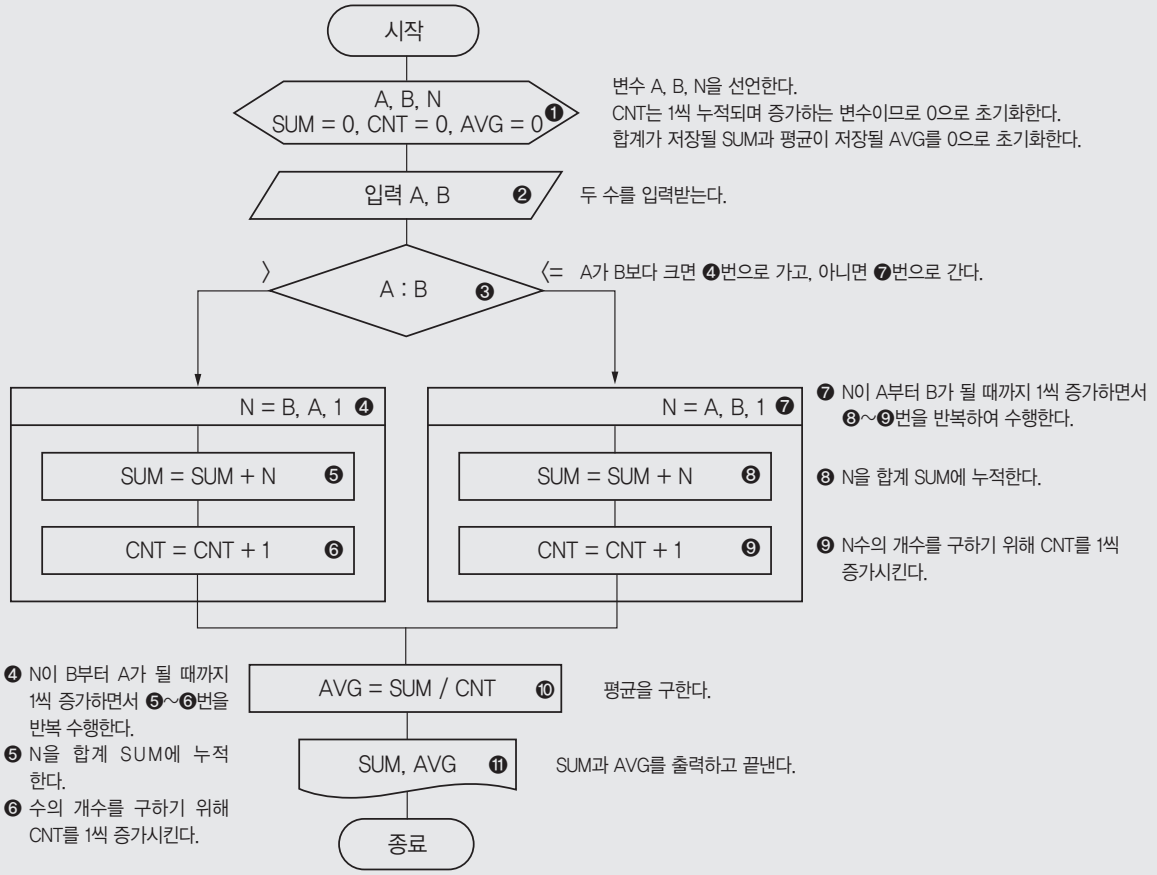


[문제 4] Section 025

① B ② B, A, 1 ③ A, B, 1 ④ N ⑤ 1 ⑥ SUM/CNT

변수설명

- A, B : 입력 받은 두 수가 저장될 변수
- N : 두 수 사이의 수를 구하기 위해 사용되는 반복 변수
- CNT : A와 B 사이에 있는 수의 개수가 저장될 변수
- SUM : A와 B 사이에 있는 수의 합계가 저장될 변수
- AVG : A와 B 사이에 있는 수의 평균이 저장될 변수

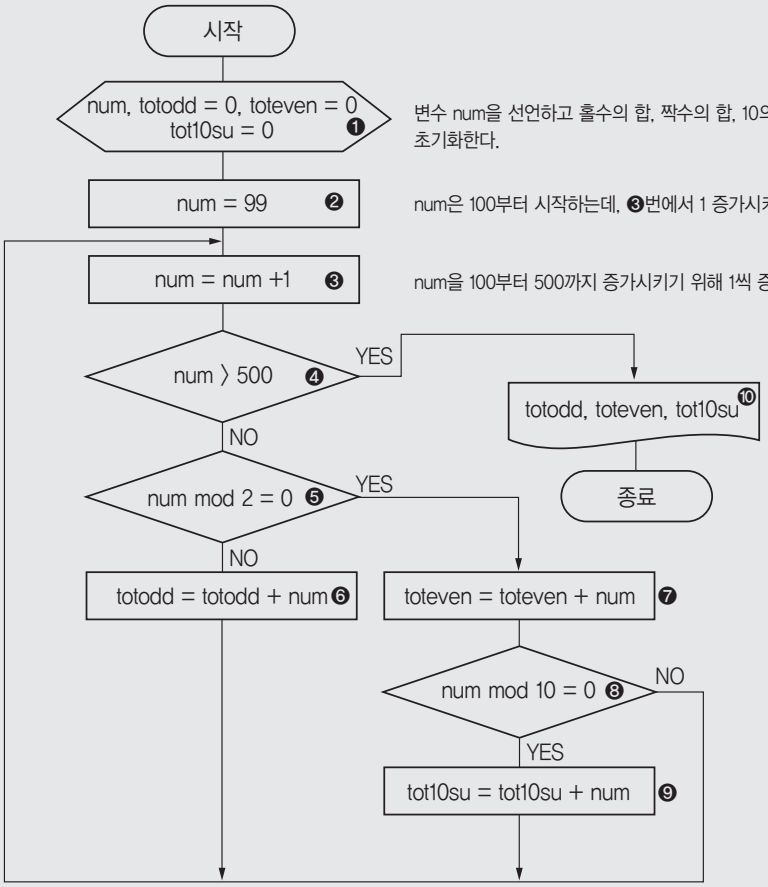


[문제 5] Section 026

① 99 ② 500 ③ 0 ④ totodd = totodd + num ⑤ toteven = toteven + num ⑥ num mod 10 = 0

변수설명

- totodd : 홀수의 합이 저장될 변수
- toteven : 짝수의 합이 저장될 변수
- tot10su : 10의 배수의 합이 저장될 변수
- num : 1씩 증가되는 숫자가 저장될 변수, 즉 num은 100, 101, …, 500까지 차례로 변경된다.



변수 num을 선언하고 홀수의 합, 짝수의 합, 10의 배수의 합이 누적될 변수를 선언하면서 각각 0으로 초기화한다.

num은 100부터 시작하는데, ❸번에서 1 증가시키므로 99로 초기화한다.

num을 100부터 500까지 증가시키기 위해 1씩 증가시킨다.

❹ num이 500보다 크면 계산 작업을 마쳤으므로 ❿번으로 가고, 아니면 다음을 수행하기 위해 ❺번으로간다.
❺ 2로 나눈 나머지가 0이면 짝수, 아니면 홀수이다. num을 2로 나눈 나머지가 0이면 ❼번으로 가고, 아니면 ❻으로 간다.

❻ 홀수의 합계를 누적한다.
❼ 짝수의 합계를 누적한다.

❽ 10으로 나눈 나머지가 0이면 10의 배수이다. 10의 배수이면 ❾번으로 가고, 아니면 바로 ❸번으로 간다.

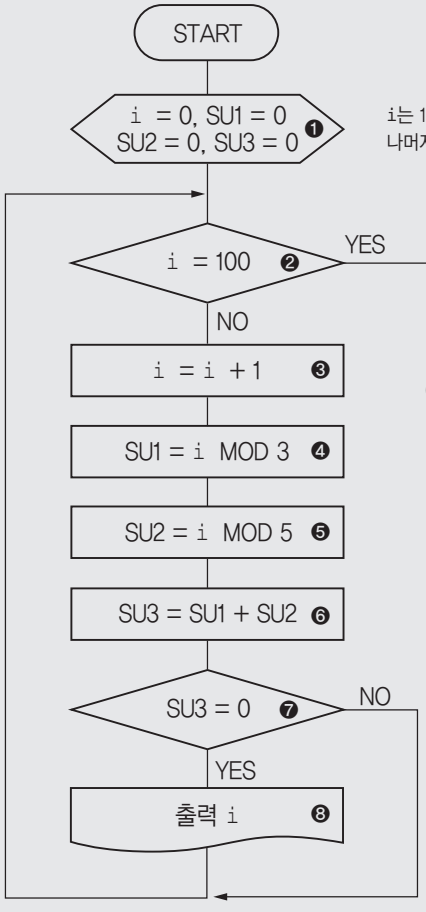
❾ 10의 배수의 합계를 누적한다.

[문제 6] Section 026

① i　② i + 1　③ SU1　④ i MOD 5　⑤ SU3　⑥ 0(Zero)

변수설명

- **i** : 1부터 100까지 1씩 증가하는 값이 저장될 변수, 즉 i는 1, 2, 3, …, 100까지 차례로 변경된다.
- SU1 : i를 3으로 나눈 나머지가 저장될 변수
- SU2 : i를 5로 나눈 나머지가 저장될 변수
- SU3 : i를 3으로 나눈 나머지와 i를 5로 나눈 나머지의 합계가 저장될 변수



i는 1씩 누적되며 증가하는 변수이므로 0으로 초기화한다. 3의 배수, 5의 배수, 3과 5의 배수인지 판별하는 나머지 값이 각각 저장될 변수 SU1, SU2, SU3를 0으로 초기화한다.

i가 100이면 모든 처리를 마친 것이므로 끝내고, 아니면 계속 처리하기 위해 ❸번으로 간다.

❸ i를 1부터 100까지 증가시키기 위해 1씩 증가시킨다.

❹ i를 3으로 나눈 나머지를 SU1에 저장한다.

❺ i를 5로 나눈 나머지를 SU2에 저장한다.

❻ SU1과 SU2를 더하여 SU3에 저장한다. SU3은 i가 3의 배수이면서 5의 배수인지를 확인하기 위한 값이다.

❼ SU3이 0이면 현재 i값이 3의 배수이면서 5의 배수이므로 ❽번으로 가고, 아니면 다음 수를 확인하기 위해 ❷번으로 간다.

❽ i를 출력하고 다음을 수행하기 위해 ❷번으로 간다.

[문제 7] Section 018

① number % i == 0

변수설명

- **i** : 계산 처리를 위한 변수
- number : 100을 기억하는 변수
- cnt : 소수의 개수가 저장될 변수

```
int main( )
{
❶   int number = 100, cnt = 0, i;
❷   for(i = 2; i < number; i++)
❸       cnt = cnt + isprime(i);
    printf("%d를 넘지 않는 소수는 %d개입니다.\n", number, cnt);
    return 0;
}
```

모든 C 프로그램은 반드시 main( ) 함수부터 시작해야 한다.

❶ 정수형 변수 number, cnt, i를 선언하고 number에는 100을 cnt에는 0을 할당한다.

❷ 반복 변수 i가 2에서 시작하여 1씩 증가하면서 number(100)보다 작은 동안 ❸번을 반복한다. 즉 ❸번 문장을 98회 반복한다.

❸ i의 값을 인수로 하여 isprime( ) 함수를 호출한 다음 돌려받은 값을 cnt와 더한 후 그 값을 다시 cnt에 저장한다. 처음에는 i가 2이므로 isprime(2)로 호출한다.

```
int isprime(int number)
{
❶   int i;
❷   for(i = 2; i < number; i++)
❸       if ( number % i == 0 )
❹           return 0;
    return 1;
}
```

isprime( ) 함수가 호출될 때 2가 전달되었으므로 number는 2이다.

❶ 정수형 지역 변수 i를 선언한다.

❷ 반복 변수 i가 2에서 시작하여 1씩 증가하면서 number보다 작은 동안 ❸번을 반복한다. 즉 number가 2이므로 ❸번 문장을 1회 반복한다.

❸ number(2)를 i(2)로 나눈 나머지가 0인지를 확인한다. 나머지가 0이므로 ❹번을 수행한다.

❹ 반환값 0을 가지고 isprime(2) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
int main()
{
    int number = 100, cnt = 0, i;
❷   for(i = 2; i < number; i++)
❶❸      cnt = cnt + isprime(i);
    printf("%d를 넘지 않는 소수는 %d개입니다.\n", number, cnt);
    return 0;
}
```

❶ isprime( ) 함수에서 돌려받은 값 0을 cnt와 더한 값 0(0+0)을 cnt에 저장한다.

❷ i의 값을 1증가 시킨 후 최종값과 비교한다. i값 3은 number(100)보다 작으므로 ❸번 문장을 수행한다.

❸ i의 값을 인수로 하여 isprime( ) 함수를 호출한 다음 돌려받은 값을 cnt와 더한 후 그 값을 다시 cnt에 저장한다. i가 3이므로 isprime(3)으로 호출한다.

```
int isprime(int number)
{
❶   int i;
❷❹ for(i = 2; i < number; i++)
❸       if ( number % i == 0 )
            return 0;
❺   return 1;
}
```

isprime( ) 함수가 호출될 때 3이 전달되었으므로 number는 3이다.

❶ 정수형 지역 변수 i를 선언한다.

❷ 반복 변수 i가 2에서 시작하여 1씩 증가하면서 number보다 작은 동안 ❸번을 반복한다. 즉 number가 3이므로 ❸번 문장을 1회 반복한다.

❸ number(3)를 i(2)로 나눈 나머지가 0인지를 확인한다. 나머지가 1이므로 제어를 ❹번으로 옮긴다.

❹ i의 값을 1증가 시킨 후 최종값과 비교한다. i값 3은 number(3)와 같으므로 반복문을 수행하지 않고 제어를 ❺번으로 옮긴다.

❺ 반환값 1을 가지고 isprime(3) 함수를 호출했던 main( ) 함수로 제어를 옮긴다.

```
int main()
{
    int number = 100, cnt = 0, i;
❷   for(i = 2; i < number; i++)
❶❸      cnt = cnt + isprime(i);
    printf("%d를 넘지 않는 소수는 %d개입니다.\n", number, cnt);
    return 0;
}
```

❶ isprime( ) 함수에서 돌려받은 값 1을 cnt와 더한 값 1(0+1)을 cnt에 저장한다.

❷ i의 값을 1증가 시킨 후 최종값과 비교한다. i값 4는 number(100)보다 작으므로 ❸번 문장을 수행한다.

❸ i의 값을 인수로 하여 isprime( ) 함수를 호출한 다음 돌려받은 값을 cnt와 더한 후 그 값을 다시 cnt에 저장한다. i가 4이므로 isprime(4)로 호출한다.

나머지 과정은 isprime( ) 함수에서 for문의 반복 변수 i를 2부터 전달받은 인수보다 작을 때까지 반복하면서 'number % i'가 0인 경우에는 0을 반환하고 그렇지 않은 경우는 1을 반환하는 과정을 반복하게 된다. 반복 과정에서 각 변수들의 값 변화는 다음 디버깅 표를 통해 확인하자.

디버깅

i (main 함수)	함수 호출	number (isprime 함수)	i (isprime 함수)	number % i	반환값 (isprime 함수)	cnt
2	isprime(2)	2	2	0	0	0
3	isprime(3)	3	2 3	1	1	1
4	isprime(4)	4	2	0	0	1
5	isprime(5)	5	2 3 4 5	3 2 1	1	2
6	isprime(6)	6	2	0	0	2
7	isprime(7)	7	2 3 4 5 6 7	5 4 3 2 1	1	4
8	isprime(8)	8	2	0	0	4
9	isprime(9)	9	2 3	1 0	0	4
10	isprime(10)	10	2	0	0	4
⋮	⋮	⋮	⋮	⋮	⋮	⋮
96	isprime(96)	96	2	0	0	
97	isprime(97)	97	2 3 ⋮ 97	1 1 ⋮	1	25
98	isprime(98)	98	2	0	0	25
99	isprime(99)	99	2 3	1 0	0	25
100						

[문제 8] Section 025

num[i]

변수설명

- num[10] : 입력 받은 숫자가 저장될 배열
- **i** : 비교 대상의 위치를 지정해 주는 변수, 즉 i는 0, 1, 2, … 9까지 차례로 변경된다.
- min : 최소값이 저장될 변수

```
#include <stdio.h>
main( )
{
❶ int num[10];                          입력 받은 숫자가 저장될 배열 num을 선언한다.
❷ int min = 9999;                       최소값이 저장될 변수 min을 선언하고 초기값으로 9999를 할당한다.
❸ int i;                                변수 i를 선언한다.
❹ for (i = 0; i < 10; i++) {            반복 변수 i가 0에서 시작하여 1씩 증가하면서 10보다 작은 동안 ❺번을 반복하여 수행한다.
❺    scanf("%d", &num[i]);              최소값을 판단할 숫자를 키보드로 입력받는다.
   }
❻ for (i = 0; i < 10; i++) {            반복 변수 i가 0에서 시작하여 1씩 증가하면서 10보다 작은 동안 ❼번을 반복하여 수행한다.
❼    if (min > num[i]) {                min이 num[i]보다 크면 ❽번으로 가고, 아니면 ❻번으로 간다.
❽       min = num[i];                   num[i]의 값을 min에 저장한다.
     }
   }
❾ printf("가장 작은 값은 %d이다.", min);   min의 값을 출력하고 끝낸다.
}
```

[문제 9] Section 020

① r == 0　② Lm++ 또는 Lm = Lm + 1

변수설명

- Lm : 자신을 제외한 약수의 합이 자신과 같은 수의 개수를 구하는 변수
- n, j, r : 계산 처리를 위한 변수
- sum : 약수들의 합을 구하는 변수
- k : 어떤 수의 모든 약수에서 자신을 제외한 약수 중 최대값

```
#include <stdio.h>

main()
{
❶ int Lm, n, j, sum, k, r;          정수형 변수 Lm, n, j, sum, k, r을 선언한다.
  Lm = 0;                           Lm은 조건에 만족하는 수의 개수가 누적되므로 Lm을 0으로 초기화한다.

❷ for (n = 4; n <= 1000; n++)       4부터 1000까지의 정수를 대상으로 하기 위해 반복 변수 n을 4부터 1000까지 1씩 증가하면서 ❸~⓫번을
                                    반복하여 수행한다.
  {
❸   sum = 0;                       약수들의 합을 구하는 변수 sum을 0으로 초기화한다.
❹   k = n / 2;                     n의 약수 중 최대값을 구하기 위해 n을 2로 나눈 몫을 k에 저장한다. C언어에서 정수형 변수는 소수점이하는 버리고 정수
                                    만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다.
❺   for (j = 1; j <= k; j++)       안쪽 반복문은 n의 약수를 구한다. j가 1에서 k가 될 때까지 1씩 증가하면서 n의 약수 인지를 계산한다.
    {
❻     r = n % j;                  n을 j로 나눈 나머지를 r에 저장한다.
❼     if (r == 0)                 r이 0이면 약수이므로 ❽번으로 가고 아니면 안쪽 for문의 시작점으로 이동하여 ❺번부터 다시 시작한다.
❽       sum += j;                 약수 j를 sum에 누적시킨다.
    }
❾   if (n == sum)                  n과 sum이 같으면 조건에 맞는 약수를 한 개 찾았으므로 ⓾번으로 가고 아니면 다음 수를 계산하기 위해 ❷번으로 간다.
    {
⓾     printf("%d ", n);           n을 출력한다.
⓫     Lm++;                       자신을 제외한 약수의 합이 자신과 같은 수의 개수를 나타내는 Lm을 1 증가시킨다
    }
  }
⓬ printf("%d", Lm);               Lm을 출력한다.
}
```

[문제 10] Section 021

① n /= k 또는 n = n / k　② c – 1

변수설명

- s[20] : 소인수가 저장될 배열
- n : 소인수로 분해하기 위해 입력 받은 숫자가 저장될 변수
- c : 소인수를 저장할 배열 s의 위치를 지정해 주는 변수
- k : 제수가 저장될 변수
- y : 소인수 분해한 결과를 출력할 때 사용할 임시 변수

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

main()
{
❶ int n, c, k, y;              정수형 변수 n, c, k, y와 소인수를 저장할 배열 s를 선언한다.
  int s[20];

  while (1)
  {
❷   scanf("%d", &n);                  소인수로 분해할 숫자를 입력 받는다.
❸   if (n < 2)                        n이 2보다 작으면 소인수 분해 과정을 모두 마친 것으로 프로그램을 종료하고 아니면 ❹번으로 간다.
      break;
❹   c = -1;                  C언어는 배열의 위치가 0부터 시작하므로 인수를 저장할 배열의 위치 c는 'c++(❽)'을 수행한 후 0이 되도록 -1로 초기화한다.
    do
    {
❺     k = 2;                 입력 받은 수 n을 2부터 나눠보기 위해 k를 2로 초기화한다.
❻     while (n%k != 0)               n을 k로 나눈 나머지가 0이 아니면 다음 수로 나눠보기 위해 ❼번으로 가고, 아니면 소인수이므로 ❽번으로 간다.
❼       k++;                 n을 다음 수로 나눠보기 위해 k를 1 증가시킨다.
❽     c++;                   소인수의 개수를 세기 위해 c를 1 증가시킨다.
❾     s[c] = k;              s[c]에 소인수 k를 저장한다.
❿     n /= k;                다음 소인수를 구하기 위해 n을 k로 나눈 몫을 다시 n에 저장한다.
⓫   } while (n != 1);                n이 1이 아니면 소인수 분해가 남은 것이므로 ❺번으로 가고, 아니면 소인수 분해 과정을 모두 마친 것이므로 ⓬번으로 간다.
⓬   if (c == 0)                      배열의 위치가 0부터 시작하므로 c가 0이면 "소수"이므로 ⓭번으로 가고, 아니면 소인수를 출력하기 위해 ⓮번으로 간다.
⓭     printf("소수");                "소수"를 출력하고 소인수로 분해할 새로운 숫자를 입력받기 위해 ❷번으로 간다.
    else
    {
⓮     for (y = 0; y <= c - 1; y++)   배열의 위치를 나타내는 반복 변수 y가 0에서 c-1이 될 때까지 1씩 증가하면서 ⓯번을 반복하여 수행한다.
⓯       printf("%dX", s[y]);         s 배열에 저장된 값을 "X" 기호와 함께 출력한다.
⓰     printf("%d", s[c]);            s 배열에 저장된 마지막 값은 "X" 기호 없이 값만 출력한다.
    }
  }
}
```

[문제 11] Section 024

① k * s[p]　② p < 8

변수설명

- s[8] : 부호를 포함한 2진수 7자리가 저장되어 있는 배열
- dec : 2진수 각 자릿수에 대한 10진수의 합계가 저장될 변수
- p : 2진수 각 자리를 지정해 주는 변수
- k : 2진수 각 자리에 대한 10진수 값이 저장될 변수

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

main()
{
❶ int dec, p, k;                    정수형 변수 dec, p, k와 부호를 포함한 2진수가 저장되어 있는 배열 s를 선언한다.
  int s[8] = { 0,0,0,1,1,1,1,1 };

❷ dec = 0;                          dec는 10진수의 합계가 누적되므로 0으로 초기화한다.
❸ p = 1;                            c언어는 배열의 위치가 0부터 시작한다. p는 부호를 제외한 두 번째 위치를 지정해야하므로 1로 초기화한다.
  do
  {
❹   k = pow(2, (7 - p));            2진수 각 자리에 해당하는 고유 자리값을 10진수로 변환한다. 첫 번째 자리는 2의 6승으로 64인데, 배열의 위치가 0부터 시작하므로 6승을 만들기 위해 지수를 (7-p)로 지정한다.
❺   dec += k * s[p];                2진수 각 자릿수에 해당하는 값을 10진수로 변환하여 dec에 누적한다.
❻   p++;                            p를 1씩 증가시킨다.
❼ } while (p < 8);                  배열의 위치가 0부터 시작하므로 p가 8보다 작으면 아직 10진수 변환이 끝나지 않은 것이므로 ❹번으로 가고, 아니면 ❽번으로 간다.

❽ if (s[0] != 0)                    배열의 위치가 0부터 시작하므로 s 배열의 첫 번째 요소는 s[0]이 된다. s[0]이 0아니면 음수로 표현된 2의 보수이므로 ❾번으로 가고, 아니면 배열 s의 값을 출력하기 위해 ⓫번으로 간다.
  {
❾   dec = 128 - dec;                2의 보수로 표현된 수치이므로 다시 2의 보수를 취해 10진수 값을 구한다. 7자리의 2진수 이므로 2의 보수를 구하려면 10000000(128)에서 해당 수치를 빼면 된다.
❿   dec *= -1;                      dec에 (-1)을 곱해 음수로 표현한다.
  }
⓫ printf("%d", dec);               dec를 출력하고 끝낸다.
}
```

[문제 12] Section 025

① a 〉 b　② sum / cnt

변수설명

- a, b : 입력 받은 두 수가 입력될 변수
- n : 두 수 사이의 수를 구하기 위해 사용되는 반복 변수
- cnt : a와 b 사이에 있는 수의 개수가 저장될 변수
- sum : a와 b 사이에 있는 수의 합계가 저장될 변수
- avg : a와 b 사이에 있는 수의 평균이 저장될 변수

```
#include 〈stdio.h〉

main()
{
❶ int a, b, n;                  정수형 변수 a, b, n을 선언한다.
  int sum = 0, cnt = 0, avg = 0;    cnt는 1씩 누적되며 증가하는 변수이므로 0으로 초기화하고 합계가 저장될 sum과 평균이 저장될 avg는 0으로 초기화한다.

❷ scanf("%d %d", &a, &b);       두 수를 입력받는다.
❸ if (a 〉 b)                    a가 b보다 크면 ❹번으로 가고, 아니면 ❼번으로 간다.
❹   for (n = b; n 〈= a; n++)     n이 b부터 a가 될 때까지 1씩 증가하면서 ❺~❻번을 반복하여 수행한다.
    {
❺     sum += n;                 n을 합계 sum에 누적한다.
❻     cnt++;                    수의 개수를 구하기 위해 cnt를 1씩 증가시킨다.
    }
  else
❼   for (n = a; n 〈= b; n++)     n이 a부터 b가 될 때까지 1씩 증가하면서 ❽~❾번을 반복하여 수행한다.
    {
❽     sum += n;                 n을 합계 sum에 누적한다.
❾     cnt++;                    수의 개수를 구하기 위해 cnt를 1씩 증가시킨다.
    }
❿ avg = sum / cnt;              평균을 구한다.
⓫ printf("%d %d", sum, avg);    sum과 avg를 출력하고 끝낸다.
}
```

[문제 13] Section 026

① 99 ② num % 2

변수설명

- totodd : 홀수의 합이 저장될 변수
- toteven : 짝수의 합이 저장될 변수
- tot10su : 10의 배수의 합이 저장될 변수
- num : 1씩 증가되는 숫자가 저장될 변수, 즉 num은 100, 101, …, 500까지 차례로 변경된다.

```
#include <stdio.h>

main()
{
❶ int num;                  정수형 변수 num을 선언하고 홀수의 합, 짝수의 합, 10의 배수의 합이 누적될 변수를 각각 0으로 초기화한다.
  int totodd = 0, toteven = 0, tot10su = 0;

❷ num = 99;                 num은 100부터 시작하는데, ❸번에서 1 증가시키므로 99로 초기화한다.
  while (1)                 while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 { } 사이의 문장을 무한 반복한다.
  {
❸   num++;                  num을 100부터 500까지 증가시키기 위해 1씩 증가시킨다.
❹   if (num > 500)                  num이 500보다 크면 계산 작업을 마쳤으므로 ❿번으로 가고, 아니면 다음을 수행하기 위해 ❺번으로 간다.
    {
❿     printf("%d %d %d", totodd, toteven, tot10su);    totodd, toteven, tot10su를 출력하고 끝낸다.
      break;                반복문(while)을 탈출하여 프로그램을 종료한다.
    }
    else
    {
❺     if (num % 2 == 0)             2로 나눈 나머지가 0이면 짝수, 아니면 홀수이다. num을 2로 나눈 나머지가 0이면 ❼번으로 가고, 아니면 ❻번으로 간다.
      {
❼       toteven += num;             짝수의 합계를 누적한다.
❽       if (num % 10 == 0)          10으로 나눈 나머지가 0이면 10의 배수이다. 10의 배수이면 ❾번으로 가고, 아니면 ❸번으로 간다.
❾         tot10su += num;           10의 배수의 합계를 누적한다.
      }
      else
❻       totodd += num;              홀수의 합계를 누적한다.

    }
  }
}
```

[문제 14] Section 026

① i % 3　② su3

변수설명

- **i** : 1부터 100까지 1씩 증가하는 값이 저장될 변수, 즉 i는 1, 2, 3, …, 100까지 차례로 변경된다.
- su1 : i를 3으로 나눈 나머지가 저장될 변수
- su2 : i를 5로 나눈 나머지가 저장될 변수
- su3 : i를 3으로 나눈 나머지와 i를 5로 나눈 나머지의 합계가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int i = 0, su1 = 0, su2 = 0, su3 = 0;
```

i는 1씩 누적되며 증가하는 변수이므로 0으로 초기화한다. 3의 배수, 5의 배수, 3과 5의 배수인지 판별하는 나머지 값이 각각 저장될 변수 su1, su2, su3를 0으로 초기화한다.

```
❷ while (i != 100)
  {
❸   i++;
❹   su1 = i % 3;
❺   su2 = i % 5;
❻   su3 = su1 + su2;
❼   if (su3 == 0)
❽     printf("%d ", i);
  }
}
```

❷ i가 100이 아니면 계속 처리하기 위해 ❸~❽번을 반복하여 수행하고, 아니면 모든 처리를 마친 것이므로 끝낸다.

❸ i를 1부터 100까지 증가시키기 위해 1씩 증가시킨다.

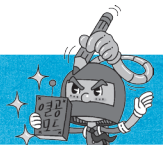
❹ i를 3으로 나눈 나머지를 su1에 저장한다.

❺ i를 5로 나눈 나머지를 su2에 저장한다.

❻ su1과 su2를 더하여 su3에 저장한다. su3은 i가 3의 배수이면서 5의 배수인지를 확인하기 위한 값이다.

❼ su3이 0이면 현재 i값이 3의 배수이면서 5의 배수이므로 ❽번으로 가고, 아니면 다음 수를 확인하기 위해 ❷번으로 간다.

❽ i를 출력하고 다음을 수행하기 위해 ❷번으로 간다.



Section **031**

[문제]

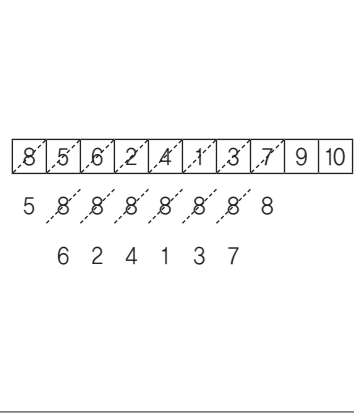
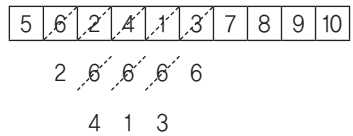
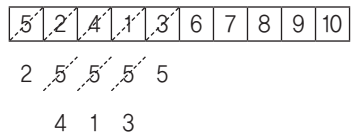
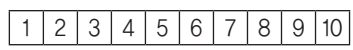
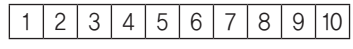
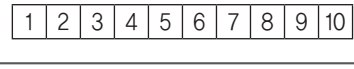
① J = i ② K = DATA[i] ③ DATA[J] = K

i	J	K	DATA[i]	DATA[J]	배열
0 1	1 2 3 4 5 6 7 8 9 10	8 5 2	8 5 5 5 2 2 2 1 1 1 1 1	5 8 6 2 5 4 1 2 3 7 9 10	8 5 6 2 4 1 3 7 9 10 5 8 5 2 2 1
2	2 3 4 5 6 7 8 9 10	8 6 5 4	8 6 6 5 5 4 4 2 2 2 2 2	6 8 5 6 4 5 2 4 3 7 9 10	1 8 6 5 4 2 3 7 9 10 6 8 6 5 4 5 4 2
3	3 4 5 6 7 8 9 10	8 6 5 4	8 6 6 5 5 4 4 3 3 3 3 3	6 8 5 6 4 5 3 4 7 9 10	1 2 8 6 5 4 3 7 9 10 6 8 6 5 4 5 4 3
. . .	. . .	. . .	. . .	. . .	. . .
8	8 9 10		8 8	9 10	1 2 3 4 5 6 7 8 9 10
9	9 10		9	10	1 2 3 4 5 6 7 8 9 10

Section **032**

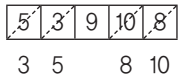
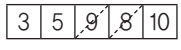
[유형 1]

① DATA[J]　② DATA[J] = DATA[J+1]　③ J 〈 (10−i)

i	J	DATA[J]	DATA[J+1]	K	10−i	배열
0 1	0 1 2 3 4 5 6 7 8 9	8 5 8 6 8 2 8 4 8 1 8 3 8 7 8 9	5 8 6 8 2 8 4 8 1 8 3 8 7 8 9 10	8 8 8 8 8 8 8 8	9 9 9 9 9 9 9 9 9 9	
2	0 1 2 3 4 5 6 7 8	5 6 2 6 4 6 1 6 3 6 7 8	6 2 6 4 6 1 6 3 6 7 8 9	6 6 6 6	8 8 8 8 8 8 8 8 8	
3	0 1 2 3 4 5 6 7	5 2 5 4 5 1 5 3 5 6 7	2 5 4 5 1 5 3 5 6 7 8	5 5 5 5	7 7 7 7 7 7 7 7	
. . .	. . .	. . .	. . .			. . .
7	0 1 2 3	1 2 3	2 3 4		3 3 3	
8	0 1 2	1 2	2 3		2 2	
9	0 1	1	2		1	

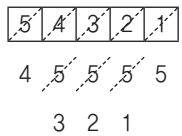
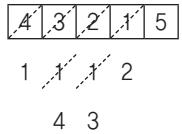
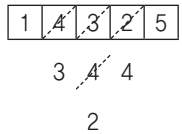
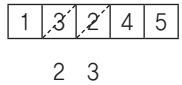
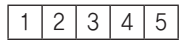
[유형 2]

① 1, 9, 1 ② CNT = CNT + 1 ③ 0

CNT	i	SW	J	5-i	DATA[J]	DATA[J+1]	K	배열
0 1 2	1	0 1 1	1 2 3 4	4	5 3 5 9 10 8	3 5 9 10 8 10	5 10	
3	2	0 1	1 2 3	3	3 5 9 8	5 9 8 9	9	
	3	0	1 2	2	3 5	5 8		

[유형 3]

① D[i] 〉 D[i+1]　② Right = Shift　③ D[i−1] 〉 D[i]　④ Left = Shift

N	Left	Right	i	D[i−1]	D[i]	D[i+1]	buf	Shift	배열
5	1	5	1 2 3 4		5 4 5 3 5 2 5 1	4 5 3 5 2 5 1 5	5 5 5 5	1 2 3 4	
		4	4 3 2	2 1 3 1 4	1 2 1 3 1 4		2 3 4	4 3 2	
	2		2 3		4 3 4 2	3 4 2 4	4 4	2 3	
		3	3	3 2	2 3		3	3	
	3								

Section **033**

① A[i]　② A[K+1] = A[K]　③ KEY

i	KEY	K	A[K]	A[K+1]	배열
2	50	1 0	80	80 50	80 50 60 20 40 50 80
3	60	2 1	80 50	80 60	50 80 60 20 40 60 80
4	20	3 2 1 0	80 60 50	80 60 50 20	50 60 80 20 40 20 50 60 80
5	40	4 3 2 1	80 60 50 20	80 60 50 40	20 50 60 80 40 40 50 60 80

Section **034**

[유형 1]

① RANK[i] = 1 ② J 〉 N ③ RANK[i] = RANK[i] + 1 ④ J = J + 1

N	i	J	JUMSU[i]	JUMSU[J]	RANK[i]	RANK 배열	출력
5	1	1 2 3 4 5 6	70 70 70 70 70	70 85 60 90 70	2 3	3 1 1 1 1	70, 85, 60, 90, 70 3, 1, 5, 1, 3
	2	1 2 3 4 5 6	85 85 85 85 85	70 85 60 90 70	2	3 2 1 1 1	
	3	1 2 3 4 5 6	60 60 60 60 60	70 85 60 90 70	2 3 4 5	3 2 5 1 1	
	4	1 2 3 4 5 6	90 90 90 90 90	70 85 60 90 70		3 2 5 1 1	
	5	1 2 3 4 5 6	70 70 70 70 70	70 85 60 90 70	2 3	3 2 5 1 3	
	6						

[유형 2]

① 1, 10, 1 ② R = 1 ③ HAP[i] 〈 HAP[J]

i	HAP[i]	J	HAP[J]	R	출력
1	150	1	150	1	1, 80, 70, 150, 2
		2	190	2	
		3	110		
2	190	1	150	1	2, 90, 100, 190, 1
		2	190		
		3	110		
3	110	1	150	1	3, 60, 50, 110, 3
		2	190	2	
		3	110	3	

Section 035

[유형 1]

① INT((L+H) / 2) ② NO ③ YES ④ J 〈 DATA[M]

J	L	H	M	DATA[M]	출력
92	1	10	5	60	92, 9
	6		8	80	
	9		9	92	

[유형 2]

① 〉 ② 〈= ③ L = M + 1

J	L	H	M	DATA[1][M]	출력
6	1	10	5	5	6, 65
	6	7	8	8	
			6	6	

Section 036

① K = K + 1 ② 〈 ③ 〉 ④ =

i	J	K	A[i]	B[J]	배열 C
1	1	0	2	1	
2	2	1	2	3	
3	3	2	3	3	
4	4	3	4	5	
5	5	4	6	5	
6		5	6	6	1 2 3 4 5 6 9 10 12 13 0
7		6	9	0	
8		7	10		
9		8	12		
		9	13		
		10	0		
		11			

Section 037

① 1 ② TOP+1 ③ TOP–1 ④ TOP–1

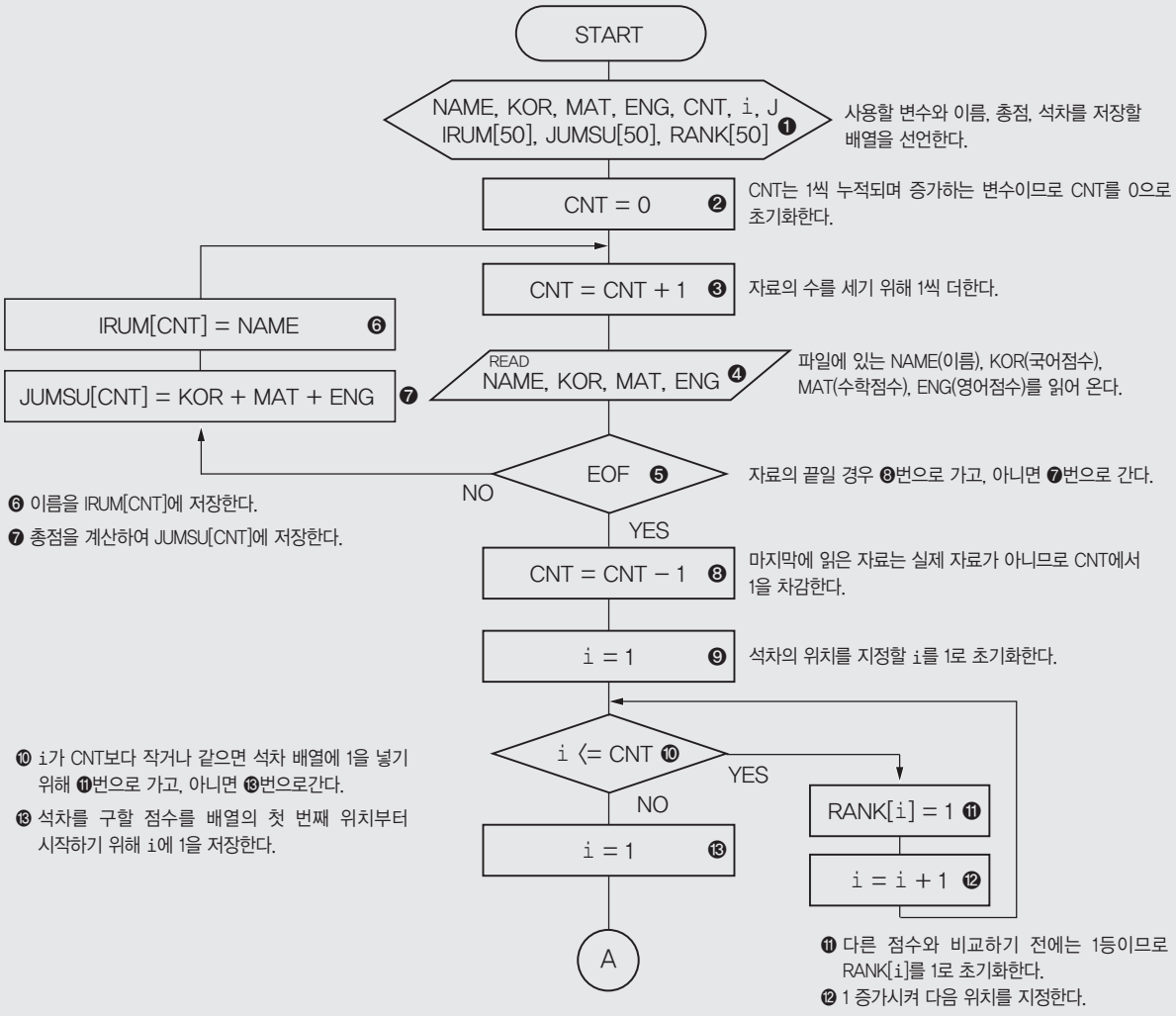
TOP	i	J	K	ii	R	STACK	출력
0	0	1	10	10	40	10	40
1	0	1	20	20	30	10 20	30
2	0	1	30	30		10 20 30	OVERFLOW
3	0	1	40	40		10 20 30 40	70
4	40	2	50	50		10 20 50 40	60
3	30	2	60	60		10 20 50 60	50
2	0	1	70	70		10 20 50 60 70	20
3	0	1	80	80			10
4	0	1					
5	–1	1					
6	5						
5	4						
	3						
	2						
	1						

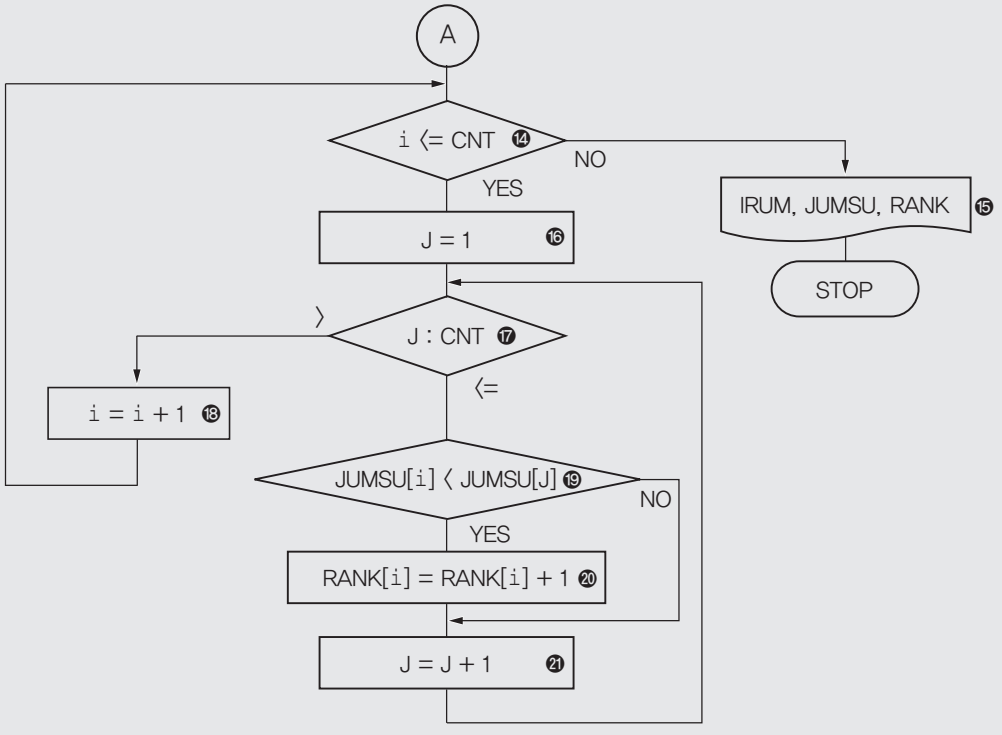
[문제 1] Section 034

① CNT ② J = 1 ③ 〉 ④ 〈= ⑤ 1

변수설명

- IRUM[50] : 이름이 저장될 배열
- RANK[50] : 석차가 저장될 변수
- JUMSU[50] : 총점이 저장될 변수, 즉 국어+수학+영어의 계산된 값이 저장된다.
- CNT : 입력 받은 데이터의 개수가 저장될 변수
- i : 석차가 저장될 배열의 위치를 지정할 변수, 비교 기준 점수의 위치를 지정해 주는 변수(회전 수)
- J : 비교 대상의 위치를 지정해 주는 변수, 각 회전에서의 비교 횟수
- NAME, KOR, MAT, ENG : 입력받은 이름, 국어, 수학, 영어 값이 저장될 변수





⑭ i가 CNT보다 작거나 같으면 다음 점수의 석차를 구하기 위해 ⑯번으로 가고, 아니면 ⑮번으로 간다.

⑮ IRUM, JUMSU, RANK 배열에 들어 있는 모든 자료를 출력하고 끝낸다.

⑯ 비교 대상을 배열의 첫 번째로 지정한다.

⑰ J가 CNT보다 크면 다음 점수의 석차를 계산하기 위해 ⑱번으로 가고, 아니면 계속 현재 점수에 대한 석차를 구하기 위해 ⑲번으로 간다.

⑱ 비교 기준 점수가 있는 위치인 i를 1부터 CNT까지 변화시키기 위해 1씩 증가시킨다.

⑲ JUMSU[i]가 JUMSU[J]보다 작으면 석차를 1 증가시켜야 하므로 ⑳번으로 가고, 아니면 ㉑번으로 간다.

⑳ 비교 기준 점수에 대한 석차 RANK[i]를 1 증가시킨다.

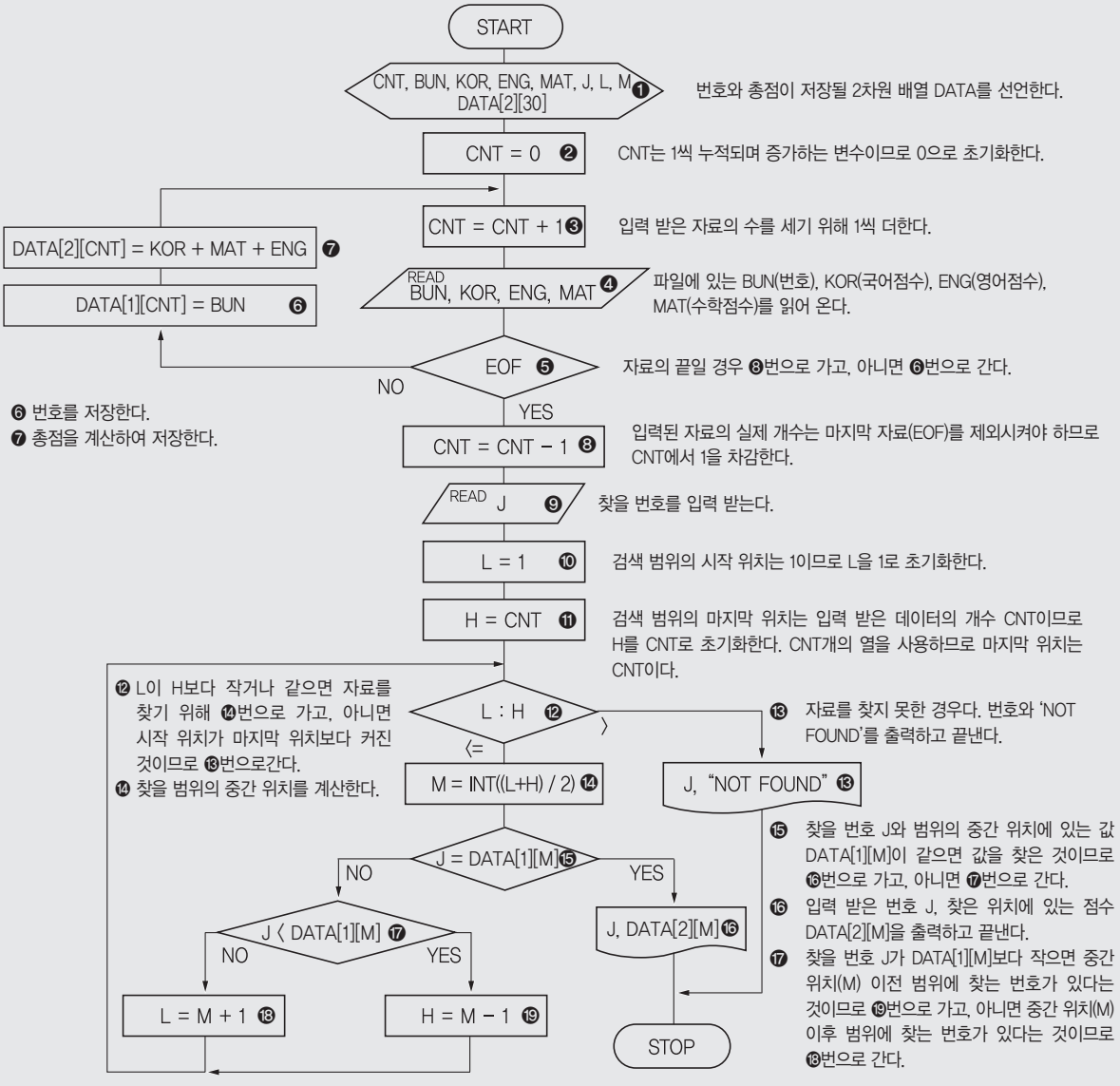
㉑ J를 1 증가시켜 다음 비교 대상 점수의 위치를 지정한다.

[문제 2] Section 035

① CNT+1 ② L+H ③ DATA[1][M] ④ M+1 ⑤ H

변수설명

- DATA[2][30] : 입력 받은 번호와 총점(국어+영어+수학)이 저장될 2차원 배열
- CNT : 입력 받은 자료의 개수가 저장될 변수
- J : 찾을 번호가 저장될 변수
- L : 검색 범위의 시작 위치를 지정해 주는 변수
- H : 검색 범위의 마지막 위치를 지정해 주는 변수
- M : 검색 범위의 중간 위치가 저장될 변수
- BUN, KOR, ENG, MAT : 입력받은 번호, 국어, 영어, 수학 값이 저장될 변수



⓲ 검색 범위의 시작 위치 L을 중간 위치보다 1칸 이후로 변경하고, ⓬번으로 간다.

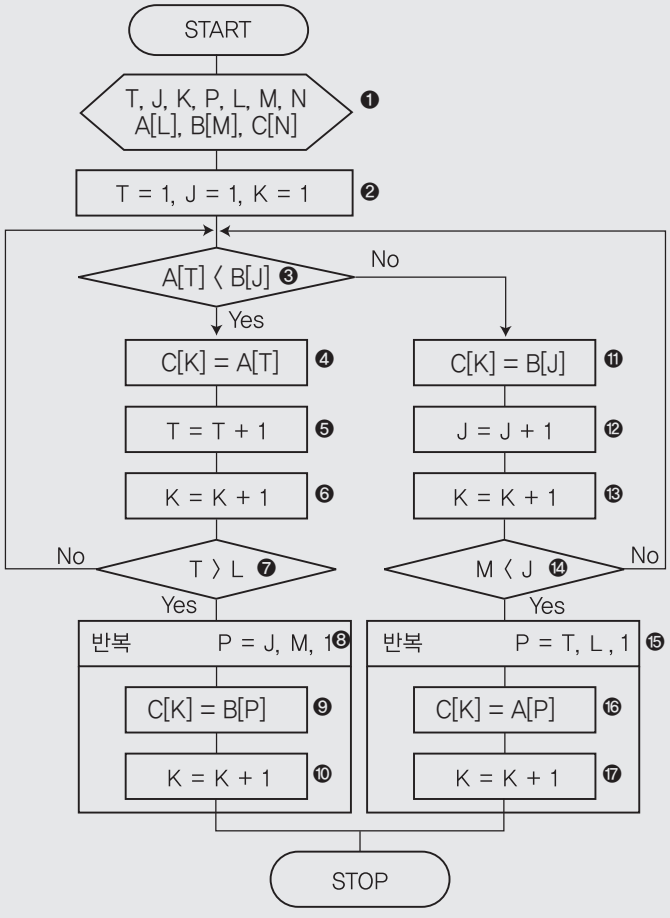
⓳ 검색 범위의 마지막 위치 H를 중간 위치보다 1칸 이전으로 변경하고, ⓬번으로 간다.

[문제 3] Section 036

① C[K] = A[T]　② C[K] = B[J]　③ C[K] = B[P]　④ L　⑤ C[K] = A[P]

변수설명

- A[L] : L개의 병합할 자료가 들어있는 배열
- B[M] : M개의 병합할 자료가 들어있는 배열
- C[N] : 2개의 배열을 병합한 자료가 들어갈 배열
- T : 배열 A의 위치를 지정해 주는 변수
- J : 배열 B의 위치를 지정해 주는 변수
- K : 배열 C의 위치를 지정해 주는 변수
- P : 배열의 위치를 지정해 주는 변수



❶ 변수 T, J, K, P, L, M, N을 선언하고 배열 A의 크기를 L, 배열 B의 크기를 M, 배열 C의 크기를 N으로 선언한다.

❷ 배열 A, B, C 각각의 첫 번째 자료를 지정하기 위해 각 배열의 위치를 지정하는 T, J, K를 1로 초기화한다.

❸ 병합할 자료 A[T]와 B[J]를 비교하여 A[T]가 작으면 배열 A의 자료를 배열 C에 넣기 위해 ❹번으로 가고, 그렇지 않으면 배열 B의 자료를 배열 C에 넣기 위해 ⓫번으로 간다.

❹ 배열 A의 자료를 배열 C에 저장한다.

❺ 배열 A의 위치를 다음 위치로 옮기기 위해 T를 1씩 증가시킨다.

❻ 배열 C의 위치를 다음 위치로 옮기기 위해 K를 1씩 증가시킨다.

❼ T가 A 배열의 크기인 L보다 크면 배열 A의 자료를 모두 처리한 것이므로 ❽번으로 가고, 아니면 병합할 자료가 남은 것이므로 ❸번으로간다.

❽ 이곳으로 온 경우는 배열 A의 자료를 모두 처리한 경우이므로 이후에는 배열 B의 자료만을 배열 C에 넣기 위해 배열 A의 자료가 모두 처리될 당시의 배열 B의 위치부터 B 배열의 크기인 M까지 배열 위치를 1씩 증가시키면서 ❾~❿을 반복한다.

❾ 배열 B의 자료를 배열 C에 저장한다.

❿ 배열 C의 위치를 다음 위치로 옮기기 위해 K를 1씩 증가시킨다.

⓫ 배열 B의 자료를 배열 C에 저장한다.

⓬ 배열 B의 위치를 다음 위치로 옮기기 위해 J를 1씩 증가시킨다.

⓭ 배열 C의 위치를 다음 위치로 옮기기 위해 K를 1씩 증가시킨다.

⓮ J가 B 배열의 크기인 M보다 크면 배열 B의 자료를 모두 처리한 것이므로 ⓯번으로 가고, 아니면 병합할 자료가 남은 것이므로 ❸번으로간다.

⓯ 이곳으로 온 경우는 배열 B의 자료를 모두 처리한 경우이므로 이후에는 배열 A의 자료만을 배열 C에 넣기 위해 배열 B의 자료가 모두 처리될 당시의 배열 A의 위치부터 A 배열의 크기인 L까지 배열 위치를 1씩 증가시키면서 ⓰~⓱을 반복한다.

⓰ 배열 A의 자료를 배열 C에 저장한다.

⓱ 배열 C의 위치를 다음 위치로 옮기기 위해 K를 1씩 증가시킨다.

[문제 4] Section 034

① <= ② i++ 또는 i = i + 1

변수설명

- irum[50][10] : 이름이 저장될 배열
- jumsu[50] : 총점이 저장될 변수, 즉 국어+수학+영어의 계산된 값이 저장된다.
- rank[50] : 석차가 저장될 변수
- cnt : 입력 받은 데이터의 개수가 저장될 변수
- **i** : 석차가 저장될 배열의 위치를 지정할 변수, 비교 기준 점수의 위치를 지정해 주는 변수(회전 수)
- **j** : 비교 대상의 위치를 지정해 주는 변수, 각 회전에서의 비교 횟수
- name[10], kor, mat, eng : 입력받은 이름, 국어, 수학, 영어 값이 저장될 변수

```
#include <stdio.h>
#include <string.h>     strcpy 쓰기 위함

main()
{
❶ FILE *inf;            • FILE : 파일을 사용할 수 있도록 만들어 놓은 포인터 형의 구조체로 stdio.h에 정의되어 있다.
                        • *inf : 파일의 시작위치를 저장할 포인터 변수로 임의의 이름을 입력하면 된다. 이 프로그램에서는 입력 파일 포인터 변수로 사용할 것이다.
❷ inf = fopen("data01.txt", "r");   'data01.txt' 파일을 읽기 모드로 연 다음 그 시작 주소를 변수 inf에 저장한다. 이제 inf에서 무엇인가 읽는다는 것은 'data01.txt' 파일의 내용을 가져온다는 의미이다.
                                    • fopen : 사용할 파일을 열어서 파일의 주소를 읽어오는 함수이다. 그대로 입력한다.
                                    • ("data01.txt", "r") : 'data01.txt' 파일을 읽기(r) 모드로 열겠다는 의미이다. 작업 폴더에 'data01.txt' 파일이 있어야 한다.
❸ char name[10];        C언어에는 문자열을 저장하는 자료형이 없기 때문에 문자 배열을 만들어서 처리한다. 배열의 크기 10은 임의로 지정하였다.
❹ int kor, mat, eng, cnt, i, j;
❺ char irum[50][10];                irum은 1차원 배열인 name의 값을 저장할 변수로 2차원 배열로 선언해야 한다.
❻ int jumsu[50], rank[50];

❼ cnt = -1;             배열의 위치가 0부터 시작하므로 배열의 첨자 cnt가 'cnt++(❾)'을 수행한 후 0이 되도록 -1로 초기화한다.
❽ while(1)              while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❾~⓭번 문장을 무한 반복한다.
  {
❾   cnt++;              자료의 수를 세기 위해 1씩 더한다.
❿   if (fscanf(inf, "%s %d %d %d", name, &kor, &mat, &eng) == EOF)   파일에서 자료를 읽어서 name, kor, mat, eng에 저장한다. 파일의 끝을 만나면 while문을 탈출한다.
                                    • fscanf(inf, "%s %d %d %d", name, &kor, &mat, &eng) : scanf()와 사용법이 비슷하다. 다른 점은 파일에서 읽기 때문에 파일 포인터를 기억하고 있는 변수를 인수로 준다는 것뿐이다. inf가 가리키고 있는 곳에서 문자열, 정수, 정수, 정수를 name, kor, mat, eng에 기억시킨다.
                                    • fscanf() == EOF : fscanf()는 파일에서 데이터를 읽다가 파일의 끝을 만나면 EOF를 반환한다. 그러니까 이 조건은 파일의 끝이면(EOF이면) break문을 실행한다.
⓫     break;            while 반복문을 탈출한다. 제어가 ⓮번으로 이동한다.
⓬   jumsu[cnt] = kor + mat + eng;   총점을 계산하여 jumsu[cnt]에 저장한다.
```

```
⑬   strcpy(irum[cnt], name);          문자 배열 name의 값을 문자 배열 irum[cnt]에 복사한다.
  }
⑭ cnt--;                            입력된 자료의 실제 개수는 마지막 자료(EOF)를 제외시켜야 하므로 cnt에서 1을 차감한다.
⑮ i = 0;                            배열의 위치가 0부터 시작하므로 석차의 위치를 지정할 i가 'i++(⑱)'을 수행한 후 1이 되도록 0으로 초기화한다.
⑯ while (i <= cnt)                  i가 cnt보다 작거나 같은 동안 ⑰~⑱번을 반복 수행한다.
  {
⑰   rank[i] = 1;                    다른 점수와 비교하기 전에는 1등이므로 rank[i]를 1로 초기화한다.
⑱   i++;                            1 증가시켜 다음 위치를 지정한다.
  }
⑲ i = 0;                            배열의 위치가 0부터 시작하므로 석차의 위치를 지정할 i가 'i++(㉒)'을 수행한 후 1이 되도록 0으로 초기화한다.
⑳ while(i <= cnt)                   i가 cnt보다 작거나 같은 동안 ㉑~㉖번을 반복 수행한다.
  {
㉑   j = 0;                          배열의 위치가 0부터 시작하므로 비교 대상을 배열의 첫 번째로 지정하기 위해 0으로 초기화한다.
㉒   while (j <= cnt)                j가 cnt보다 작거나 같은 동안 ㉓~㉕번을 반복 수행한다.
    {
㉓     if (jumsu[i] < jumsu[j])      jumsu[i]가 jumsu[j]보다 작으면 석차를 1 증가시켜야 하므로 ㉔번으로 가고, 아니면 ㉕번으로 간다.
㉔       rank[i]++;                 비교 기준 점수에 대한 석차 rank[i]를 1 증가시킨다.
㉕     j++;                         j를 1 증가시켜 다음 비교 대상 점수의 위치를 지정한다.
    }
㉖   i++;                            비교 기준 점수가 있는 위치인 i를 0부터 cnt까지 변화시키기 위해 1씩 증가시킨다.
  }
㉗ for (int x = 0; x <= cnt; x++)   x가 0에서 시작하여 cnt보다 작거나 같은 동안 irum, jumsu, rank 배열에 들어 있는 모든 자료를 출력하고 끝낸다.
㉘   printf("%s %d %d\n", irum[x], jumsu[x], rank[x]);
㉙ fclose(inf);                      파일 변수 inf를 닫는다. 읽기 형식의 파일을 닫을 경우 메모리가 확보되고, 쓰기 형식의 파일을 닫을 경우 확실하게 저장하기 위함이다.
                                    • fclose : 파일을 닫는 함수이다. 그대로 적어준다.
                                    • (inf) : 파일 변수를 입력한다.
}
```

[문제 5] Section 035

① == ② m – 1

변수설명

- data[2][30] : 입력 받은 번호와 총점(국어+영어+수학)이 저장될 2차원 배열
- cnt : 입력 받은 자료의 개수가 저장될 변수
- **j** : 찾을 번호가 저장될 변수
- L : 검색 범위의 시작 위치를 지정해 주는 변수
- h : 검색 범위의 마지막 위치를 지정해 주는 변수
- m : 검색 범위의 중간 위치가 저장될 변수
- bun, kor, eng, mat : 입력받은 번호, 국어, 영어, 수학 값이 저장될 변수

```
#include <stdio.h>

main()
{
❶ FILE *inf;
❷ inf = fopen("data02.txt", "r");
❸ int cnt, j, L, h, m, bun, kor, eng, mat;
❹ int data[2][30];

❺ cnt = -1;
❻ while (1)
  {
❼   cnt++;
❽   if (fscanf(inf, "%d %d %d %d", &bun, &kor, &eng, &mat) == EOF)
❾     break;
❿   data[0][cnt] = bun;
⓫   data[1][cnt] = kor + mat + eng;
  }
⓬ cnt--;
⓭ scanf("%d", &j);
```

- ❶ FILE *inf; : • FILE : 파일을 사용할 수 있도록 만들어 놓은 포인터 형의 구조체로 stdio.h에 정의되어 있다. • *inf : 파일의 시작위치를 저장할 포인터 변수로 임의의 이름을 입력하면 된다. 이 프로그램에서는 입력 파일 포인터 변수로 사용할 것이다.
- ❷ inf = fopen("data02.txt", "r"); : 'data02.txt' 파일을 읽기 모드로 연 다음 그 시작 주소를 변수 inf에 저장한다. 이제 inf에서 무엇인가 읽는다는 것은 'data02.txt' 파일의 내용을 가져온다는 의미이다. • fopen : 사용할 파일을 열어서 파일의 주소를 읽어오는 함수이다. 그대로 입력한다. • ("data02.txt", "r") : 'data02.txt' 파일을 읽기(r) 모드로 열겠다는 의미이다. 작업 폴더에 'data02.txt' 파일이 있어야 한다.
- ❸ int cnt, j, L, h, m, bun, kor, eng, mat; / ❹ int data[2][30]; :
- ❺ cnt = -1; : 배열의 위치가 0부터 시작하므로 배열의 첨자 cnt가 'cnt++(❼)'을 수행한 후 0이 되도록 -1로 초기화한다.
- ❻ while (1) : while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❼~⓫번 문장을 무한 반복한다.
- ❼ cnt++; : 자료의 수를 세기 위해 1씩 더한다.
- ❽ if (fscanf(inf, "%d %d %d %d", &bun, &kor, &eng, &mat) == EOF) : 파일에서 자료를 읽어서 bun, kor, eng, mat에 저장한다. 파일의 끝을 만나면 while문을 탈출한다. • fscanf(inf, "%d %d %d %d", &bun, &kor, &eng, &mat) : scanf()와 사용법이 비슷하다. 다른 점은 파일에서 읽기 때문에 파일 포인터를 기억하고 있는 변수를 인수로 준다는 것뿐이다. inf가 가리키고 있는 곳에서 정수 4개를 bun, kor, eng, mat에 기억시킨다. • fscanf() == EOF : fscanf()는 파일에서 데이터를 읽다가 파일의 끝을 만나면 EOF를 반환한다. 그러니까 이 조건은 파일의 끝이면(EOF이면) break문을 실행한다.
- ❾ break; : while 반복문을 탈출한다. 제어가 ⓬번으로 이동한다.
- ❿ data[0][cnt] = bun; : 번호를 저장한다.
- ⓫ data[1][cnt] = kor + mat + eng; : 총점을 계산하여 저장한다.
- ⓬ cnt--; : 입력된 자료의 실제 개수는 마지막 자료(EOF)를 제외시켜야 하므로 cnt에서 1을 차감한다.
- ⓭ scanf("%d", &j); : 찾을 번호 j를 입력받는다.

```
⑭ L = 0;                    배열의 위치가 0부터 시작하므로 검색 범위의 시작 위치인 L을 0으로 초기화한다.
⑮ h = cnt;                  검색 범위의 마지막 위치는 입력 받은 데이터의 개수 cnt이므로 h를 cnt로 초기화한다.
⑯ while (1)                 while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ⑰~㉓번을 무한 반복한다.
   {
⑰    if (L <= h)            L이 h보다 작거나 같으면 자료를 찾기 위해 ⑱번으로 가고, 아니면 시작 위치가 마지막 위치보다 커진 것이므로 ㉗번으로 간다.
     {
⑱      m = (L + h) / 2;          C언어에서 정수형 변수는 소수점이하는 버리고 정수만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다.
⑲      if (j == data[0][m])      찾을 번호 j와 범위의 중간 위치에 있는 값 data[0][m]이 같으면 값을 찾은 것이므로 ⑳번으로 가고, 아니면 ㉒
                                 번으로 간다.
       {
⑳        printf("%d %d", j, data[1][m]);   입력 받은 j와 찾은 위치에 있는 점수 data[1][m]을 출력한다.
㉑        break;                    while 반복문을 탈출한다. 제어가 ㉙번으로 이동한다.
       }
㉒     if (j < data[0][m])       찾을 번호 j가 data[0][m]보다 작으면 중간 위치(m) 이전 범위에 찾는 번호가 있다는 것이므로 ㉓번으로 가고,
                                 아니면 중간 위치(m) 이후 범위에 찾는 번호가 있다는 것이므로 ㉕번으로 간다.
㉓       h = m - 1;              검색 범위의 마지막 위치 h를 중간 위치보다 1칸 이전으로 변경하고 ⑯번으로 간다.
㉔     else
㉕       L = m + 1;              검색 범위의 시작 위치 L을 중간 위치보다 1칸 이후로 변경하고 ⑯번으로 간다.
     }
㉖   else
     {
㉗     printf("%d NOT FOUND", j);   자료를 찾지 못한 경우다. 번호와 'NOT FOUND'를 출력한다.
㉘     break;                  while 반복문을 탈출한다. 제어가 ㉙번으로 이동한다.
     }
   }
㉙ fclose(inf);              파일 변수 inf를 닫는다. 읽기 형식의 파일을 닫을 경우 메모리가 확보되고, 쓰기 형식의 파일을 닫을 경우 확실하게 저장하기
                            위함이다.
                            • fclose : 파일을 닫는 함수이다. 그대로 적어준다.
                            • (inf) : 파일 변수를 입력한다.
}
```

[문제 6] Section 036

① t ② j

변수설명

- a[L] : L개의 병합할 자료가 들어있는 배열
- b[m] : m개의 병합할 자료가 들어있는 배열
- c[n] : 2개의 배열을 병합한 자료가 들어갈 배열
- t : 배열 a의 위치를 지정해 주는 변수
- **j** : 배열 b의 위치를 지정해 주는 변수
- k : 배열 c의 위치를 지정해 주는 변수
- p : 배열의 위치를 지정해 주는 변수

```
#include <stdio.h>

main()
{
❶ int t, j, k, p;
   int L = 5, m = 9, n = 20;          L, m, n은 배열 a, b, c의 크기로, 각각 5, 9, 20으로 가정한다.
   int a[5] = { 1,3,5,6,7 };          배열 a에 5개 데이터가 저장되어 있다고 가정한다.
   int b[9] = { 2,3,5,8,9,10,12,13,14 };   배열 b에 9개의 데이터가 저장되어 있다고 가정한다.
   int c[20];

❷ t = j = k = 0;                      배열의 위치는 0부터 시작하므로 첫 번째 자료를 지정하기 위한 첨자 변수들을 0으로 초기화한다.
❸ while (1)                           while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❹~❷번을 무한 반복한다.
   {
❹   if (a[t] < b[j])                  a[t]가 b[j]보다 작으면 배열 a의 자료를 배열 c에 넣기 위해 ❺번으로 가고, 아니면 배열 b의 자료를 배열 c에 넣기 위해 ⓮번으로 간다.
     {
❺     c[k] = a[t];                    배열 a의 자료를 배열 c에 저장한다.
❻     t++;                            배열 a의 위치를 다음 위치로 옮기기 위해 t를 1씩 증가시킨다.
❼     k++;                            배열 c의 위치를 다음 위치로 옮기기 위해 k를 1씩 증가시킨다.
❽     if (t > L - 1)                  배열의 위치가 0부터 시작하므로 배열 a의 크기는 'L-1'이다. t가 배열의 크기보다 크면 배열 a의 자료를 모두 처리한 것이므로 ❾번으로 가고, 아니면 병합할 자료가 남은 것이므로 ❹번으로 간다.
       {
❾        for (p = j; p <= m-1; p++)   배열의 위치가 0부터 시작하므로 배열 b의 크기는 'm-1'이다. 이곳으로 온 경우는 배열 a의 자료를 모두 처리한 경우이므로 이후에는 배열 b의 자료만을 배열 c에 넣기 위해 배열 a의 자료가 모두 처리될 당시의 배열 b의 위치부터 b 배열의 크기인 'm-1'까지 배열 위치를 1씩 증가시키면서 ❿~⓫번을 반복 수행한다.
         {
❿          c[k] = b[p];               배열 b의 자료를 배열 c에 저장한다.
⓫          k++;                       배열 c의 위치를 다음 위치로 옮기기 위해 k를 1씩 증가시킨다.
         }
⓬        break;                       while 반복문을 탈출한다. 제어가 ❷번으로 이동한다.
       }
     }
⓭   else
     {
⓮     c[k] = b[j];                    배열 b의 자료를 배열 c에 저장한다.
⓯     j++;                            배열 b의 위치를 다음 위치로 옮기기 위해 j를 1씩 증가시킨다.
```

```
⑯      k++;                  배열 c의 위치를 다음 위치로 옮기기 위해 k를 1씩 증가시킨다.
⑰      if (m-1 < j)          배열의 위치가 0부터 시작하므로 배열 b의 크기는 'm-1'이다. j가 배열의 크기보다 크면 배열 b의 자료를 모두 처리한 것이므로
                             ⑱번으로 가고, 아니면 병합할 자료가 남은 것이므로 ❹번으로 간다.
       {
⑱        for (p = t; p <= L-1; p++)   배열의 위치가 0부터 시작하므로 배열 a의 크기는 'L-1'이다. 이곳으로 온 경우는 배열 b의 자료를 모두 처리
                                      한 경우이므로 이후에는 배열 a의 자료만을 배열 c에 넣기 위해 배열 b의 자료가 모두 처리될 당시의 배열 a의
                                      위치부터 a 배열의 크기인 'L-1'까지 배열 위치를 1씩 증가시키면서 ⑲~⑳번을 반복 수행한다.
          {
⑲           c[k] = a[p];          배열 a의 자료를 배열 c에 저장한다.
⑳           k++;                  배열 c의 위치를 다음 위치로 옮기기 위해 k를 1씩 증가시킨다.
          }
㉑        break;                   while 반복문을 탈출한다. 제어가 ㉒번으로 이동한다.
       }
    }
  }
} ㉒
```

Section **038**

[유형 1]

① 1, 5, 1 ② 1, 5, 1 ③ A[i][J] = K

i	J	K	배열 A
1	1	0	
	2	1	
	3	2	
	4	3	
	5	4	
		5	
2	1	6	
	2	7	
	3	8	
	4	9	
	5	10	
3	1	11	
	2	12	
	3	13	
	4	14	
	5	15	
4	1	16	
	2	17	
	3	18	
	4	19	
	5	20	
5	1	21	
	2	22	
	3	23	
	4	24	
	5	25	

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

[유형 2]

① i = 1, 5, 1 ② J = 1, 5, 1

i	J	K	배열 A
1	1	0	
	2	1	
	3	2	
	4	3	
	5	4	
		5	
2	1	6	
	2	7	
	3	8	
	4	9	
	5	10	
3	1	11	
	2	12	
	3	13	
	4	14	
	5	15	
4	1	16	
	2	17	
	3	18	
	4	19	
	5	20	
5	1	21	
	2	22	
	3	23	
	4	24	
	5	25	

1	6	11	16	21
2	7	12	17	22
3	8	13	18	23
4	9	14	19	24
5	10	15	20	25

Section **039**

[유형 1]

① J = 1, i, 1　② A[i][J] = K

<table>
<tr><th>i</th><th>J</th><th>K</th><th>배열 A</th></tr>
<tr><td>1</td><td>1
</td><td>0
1</td><td rowspan="5"><table>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>2</td><td>3</td><td></td><td></td><td></td></tr>
<tr><td>4</td><td>5</td><td>6</td><td></td><td></td></tr>
<tr><td>7</td><td>8</td><td>9</td><td>10</td><td></td></tr>
<tr><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr>
</table></td></tr>
<tr><td>2</td><td>1
2</td><td>2
3</td></tr>
<tr><td>3</td><td>1
2
3</td><td>4
5
6</td></tr>
<tr><td>4</td><td>1
2
3
4</td><td>7
8
9
10</td></tr>
<tr><td>5</td><td>1
2
3
4
5</td><td>11
12
13
14
15</td></tr>
</table>

[유형 2]

① J = 6–i, 5, 1

i	J	K	배열 A
1	5	0 1	
2	4 5	2 3	
3	3 4 5	4 5 6	
4	2 3 4 5	7 8 9 10	
5	1 2 3 4 5	11 12 13 14 15	

				1
			2	3
		4	5	6
	7	8	9	10
11	12	13	14	15

[유형 3]

① J = i, 1, –1

<table>
<tr><th>i</th><th>J</th><th>K</th><th>배열 A</th></tr>
<tr><td>1</td><td>1
</td><td>0
1</td><td rowspan="6"><table>
<tr><td>1</td><td></td><td></td><td></td><td></td></tr>
<tr><td>3</td><td>2</td><td></td><td></td><td></td></tr>
<tr><td>6</td><td>5</td><td>4</td><td></td><td></td></tr>
<tr><td>10</td><td>9</td><td>8</td><td>7</td><td></td></tr>
<tr><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td></tr>
</table></td></tr>
<tr><td>2</td><td>2
1</td><td>2
3</td></tr>
<tr><td>3</td><td>3
2
1</td><td>4
5
6</td></tr>
<tr><td>4</td><td>4
3
2
1</td><td>7
8
9
10</td></tr>
<tr><td>5</td><td>5
4
3
2
1</td><td>11
12
13
14
15</td></tr>
</table>

Section **040**

① J = L, M, N ② L = M ③ N = N × (–1)

P	L	M	N	i	J	K	배열 A
	1	5	1	1	1 2 3 4 5	0 1 2 3 4 5	
1	5	1	–1	2	5 4 3 2 1	6 7 8 9 10	
5	1	5	1	3	1 2 3 4 5	11 12 13 14 15	
1	5	1	–1	4	5 4 3 2 1	16 17 18 19 20	
5	1	5	1	5	1 2 3 4 5	21 22 23 24 25	
1	5	1	–1				

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

Section **041**

① J = S, E, 1 ② i >= 3 ③ S = S - 1 ④ E = E + 1

S	E	i	J	K	배열 A
3	3	1	3	0 1	
2	4	2	2 3 4	2 3 4	
1	5	3	1 2 3 4 5	5 6 7 8 9	
2	4	4	2 3 4	10 11 12	
3	3	5	3	13	
4	2				

		1		
	2	3	4	
5	6	7	8	9
	10	11	12	
		13		

Section 042

① i = 1, M, 1 ② J = i, L, 1 ③ i = M+1, X, 1 ④ J = L, i, 1

X	M	i	L	J	K	배열 A
5	3	1	6	1	0	
			5	2	1	
				3	2	
				4	3	
				5	4	
					5	
		2	4	2	6	
				3	7	
				4	8	
		3	3	3	9	
		4	2	2	10	
				3	11	
				4	12	
		5	1	1	13	
				2	14	
				3	15	
				4	16	
				5	17	

1	2	3	4	5
	6	7	8	
		9		
	10	11	12	
13	14	15	16	17

Section **043**

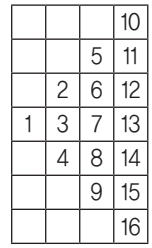
① M = INT(X/2)+1 ② L = (X+1) − i ③ L = i

X	M	i	L	J	K	배열 A
7	4	1	7	1 2 3 4 5 6 7	0 1 2 3 4 5 6 7	
		2	6	1 2 3 4 5 6	8 9 10 11 12 13	
		3	5	1 2 3 4 5	14 15 16 17 18	
		4	4	1 2 3 4	19 20 21 22	
		5	5	1 2 3 4 5	23 24 25 26 27	
		6	6	1 2 3 4 5 6	28 29 30 31 32 33	
		7	7	1 2 3 4 5 6 7	34 35 36 37 38 39 40	

1	2	3	4	5	6	7
8	9	10	11	12	13	
14	15	16	17	18		
19	20	21	22			
23	24	25	26	27		
28	29	30	31	32	33	
34	35	36	37	38	39	40

Section 044

① (COL+1) – J　② J + (COL–1)

COL	ROW	J	L	E	i	K	배열 A
4	7	1	4	4	4	0 1	
		2	3	5	3 4 5	2 3 4	
		3	2	6	2 3 4 5 6	5 6 7 8 9	
		4	1	7	1 2 3 4 5 6 7	10 11 12 13 14 15 16	

Section **045**

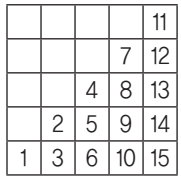
① A[i][J] = K　② B[J][6-i]

배열 A

i	J	K	배열 A
5	5	0	
		1	
4	4	2	
	5	3	
3	3	4	
	4	5	
	5	6	
2	2	7	
	3	8	
	4	9	
	5	10	
1	1	11	
	2	12	
	3	13	
	4	14	
	5	15	

11	12	13	14	15
	7	8	9	10
		4	5	6
			2	3
				1

배열 B

i	J	A[i][J]	6-i	배열 B
1	1	11	5	
	2	12	5	
	3	13	5	
	4	14	5	
	5	15	5	
2	1		4	
	2	7	4	
	3	8	4	
	4	9	4	
	5	10	4	
3	1		3	
	2		3	
	3	4	3	
	4	5	3	
	5	6	3	
4	1		2	
	2		2	
	3		2	
	4	2	2	
	5	3	2	
5	1		1	
	2		1	
	3		1	
	4		1	
	5	1	1	

Section 046

① C = 1 ② F = 5 ③ F = F – 1 ④ C = C × (–1)

C	F	i	J	N	K	배열 A
1	5	1	0 1 2 3 4 5	1 2 3 4 5	0 1 2 3 4 5	
	4	2 3 4 5		1 2 3 4	6 7 8 9	
–1			4 3 2 1	1 2 3 4	10 11 12 13	
	3	4 3 2		1 2 3	14 15 16	
1			2 3 4	1 2 3	17 18 19	
	2	3 4		1 2	20 21	
–1			3 2	1 2	22 23	
	1	3		1	24	
1	0		3	1	25	

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

Section 047

① K = i − J ② K 〉 5

i	J	K	L	배열 A
2	1 2 3 4 5	1 0 −1 −2 −3	0 1	
3	1 2 3 4 5	2 1 0 −1 −2	2 3	
4	1 2 3 4 5	3 2 1 0 −1	4 5 6	
5	1 2 3 4 5	4 3 2 1 0	7 8 9 10	
6	1 2 3 4 5	5 4 3 2 1	11 12 13 14 15	
7	1 2 3 4 5	6 5 4 3 2	16 17 18 19	
8	1 2 3 4 5	7 6 5 4 3	20 21 22	
9	1 2 3 4 5	8 7 6 5 4	23 24	
10	1 2 3 4 5	9 8 7 6 5	25	

1	2	4	7	11
3	5	8	12	16
6	9	13	17	20
10	14	18	21	23
15	19	22	24	25

Section **048**

① i = i + 1 ② i = 5 ③ J = 1

i	J	K	NMG	배열 A
1	3	1	1	
0	4	2	2	
5	5	3	3	
4	6	4	4	
3	1	5	0	
2	2	6	1	
3	3	7	2	
2	4	8	3	
1	5	9	4	
0	6	10	0	
5	1	11	1	
4	2	12	2	
5	3	13	3	
4	4	14	4	
3	5	15	0	
2	6	16	1	
1	1	17	2	
2	2	18	3	
1	3	19	4	
0	4	20	0	
5	5	21	1	
4	6	22	2	
3	1	23	3	
4	2	24	4	
3	3	25	0	
2				
1				
0				
5				
6				

17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

Section **049**

① B[L][M]　② L = L + 1　③ M = 0

배열 A

ROW	COL	i	J	K	배열 A
5	3	1	1 2 3	0 1 2 3	
		2	1 2 3	4 5 6	
		3	1 2 3	7 8 9	
		4	1 2 3	10 11 12	
		5	1 2 3	13 14 15	

1	2	3	
4	5	6	
7	8	9	
10	11	12	
13	14	15	

배열 B

i	J	L	M	A[i][J]	배열 B
1	1 2 3	1	0 1 2 3	1 2 3	
2	1 2 3	2	4 5 0 1	4 5 6	
3	1 2 3		2 3 4	7 8 9	
4	1 2 3	3	5 0 1 2	10 11 12	
5	1 2 3	4	3 4 5 0	13 14 15	

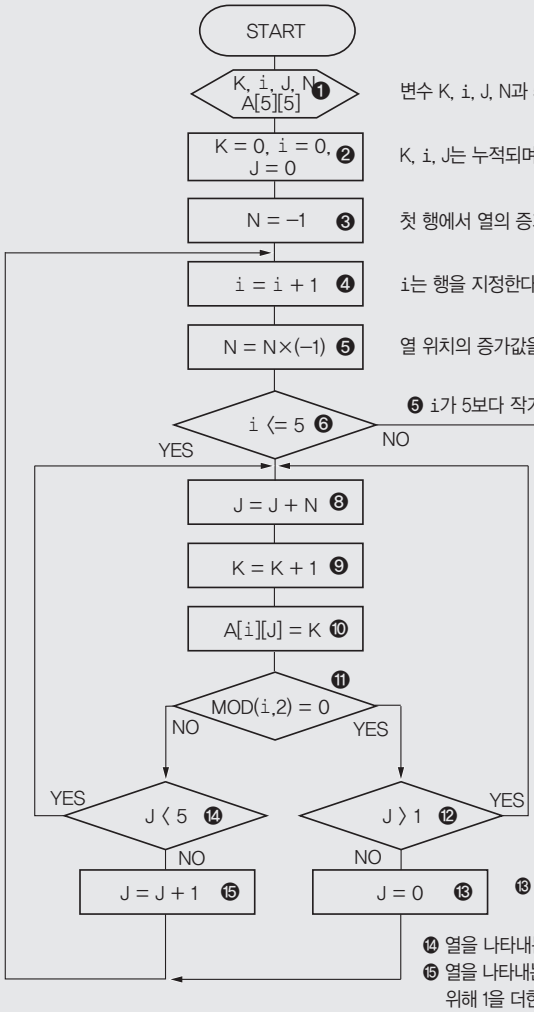
1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

[문제 1] Section 040

① N×(–1) ② J + N ③ K + 1 ④ J + 1 ⑤ J 〉 1

변수설명

- A[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- K : 1씩 증가되는 숫자가 저장될 변수, 즉 K는 1, 2, 3, …, 25까지 차례로 변경된다.
- i : 배열의 행 위치를 지정해 주는 변수
- J : 배열의 열 위치를 지정해 주는 변수
- N : 열 위치의 증가 혹은 감소 여부를 지정해 주는 변수



변수 K, i, J, N과 5행 5열의 2차원 배열 A를 선언한다.

K, i, J는 누적되며 증가하는 변수이므로 0으로 초기화한다.

첫 행에서 열의 증가 값은 1이지만 'N=N×(–1)'(❺)을 실행한 후 1이되도록 –1로 초기화한다.

i는 행을 지정한다. 행을 지정하는 i를 1에서 5까지 증가시키기 위해 1을 더한다.

열 위치의 증가값을 1 또는 –1로 변경하기 위해 N에 –1을 곱한다.

❺ i가 5보다 작거나 같으면 ❽번으로 가고, 아니면 모두 수행했으므로 ❼번으로 간다.

A 배열의 내용을 출력하고 끝낸다.

❽ J는 열을 지정한다. 열을 지정하는 J를 1에서 5 또는 5에서 1까지 증가시키기 위해 J에 N을 누적한다.
❾ K를 1에서 25까지 증가시키기 위해 1을 더한다.
❿ A 배열의 i행 J열(A[i][J])에 K를 저장한다.
⓫ i가 짝수이면 ⓬번으로 가고, 아니면 ⓮번으로 간다.
⓬ 열을 나타내는 J가 1보다 크면 ❽번으로 가고, 아니면 ⓭번으로 간다.

⓭ 열을 나타내는 J가 1이 되었을 경우이다. 다음 행의 열 J는 1에서 5까지 변해야므로 J를 0으로 초기화한다.

⓮ 열을 나타내는 J가 5보다 작으면 ❽번으로 가고, 아니면 ⓯번으로 간다.
⓯ 열을 나타내는 J가 5가 되었을 경우이다. 다음 행의 열 J는 5에서 1까지 변해야 하므로 6을 만들기 위해 1을 더한다.

디버깅

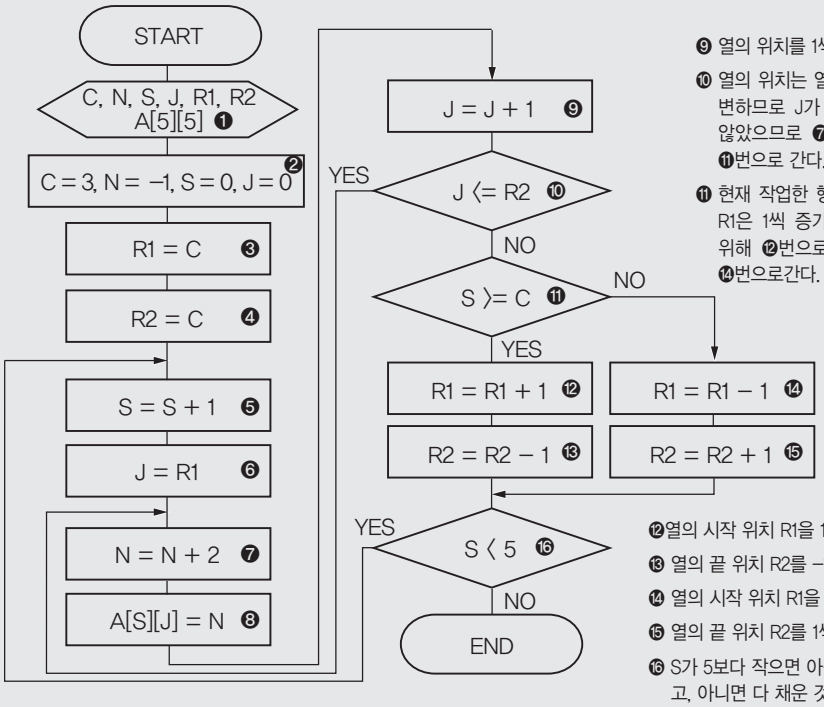
K	N	i	J	A[i][J]	MOD(i,2)	출력
0	–1	0	0	1	1	
1	1	1	1	2	1	
2	–1	2	2	3	1	
3	1	3	3	4	1	
4	⋮	⋮	4	5	1	
5			5	6	0	
6			6	7	0	1 2 3 4 5 10 9 8 7 6 11 12 …
7			5	8	0	
8			4	9	0	
9			3	10	0	
10			2	11	1	
11			1	12	1	
12			0	⋮	⋮	
⋮			1			
			2			
			⋮			

[문제 2] Section 042

① N+2 ② A[S][J] ③ R2 ④ R2–1 ⑤ R1–1

변수설명

- C : 행의 중간 위치가 저장될 변수
- N : 2씩 증가되는 숫자가 저장될 변수, 즉 N은 1, 3, 5, …, 23, 25까지 차례로 변경된다.
- S : 배열의 행 위치를 지정해 주는 변수
- J : 배열의 열 위치를 지정해 주는 변수
- R1 : 열의 시작 위치가 저장될 변수, 즉 R1이 2이면 2열부터 데이터를 저장한다.
- R2 : 열의 끝 위치가 저장될 변수, 즉 R2가 4이면 4열까지 데이터를 저장한다.



❾ 열의 위치를 1씩 증가시킨다.

❿ 열의 위치는 열의 시작 위치인 R1부터 열의 끝 위치인 R2까지 변하므로 J가 R2보다 작거나 같으면 현재 행을 다 채우지 않았으므로 ❼번으로 가고, 아니면 다음 행을 채우기 위해 ⓫번으로 간다.

⓫ 현재 작업한 행(S)이 3(C)보다 크거나 같으면 열의 시작 위치 R1은 1씩 증가시키고, 열의 끝 위치 R2는 –1씩 증가시키기 위해 ⓬번으로 가고, 아니면 반대의 작업을 수행하기 위해 ⓮번으로간다.

⓬열의 시작 위치 R1을 1씩 증가시킨다.

⓭ 열의 끝 위치 R2를 –1씩 증가시킨다.

⓮ 열의 시작 위치 R1을 –1씩 증가시킨다.

⓯ 열의 끝 위치 R2를 1씩 증가시킨다.

⓰ S가 5보다 작으면 아직 채워야 할 행이 남았으므로 ❺번으로가고, 아니면 다 채운 것이므로 끝낸다.

❶ 변수 C, N, S, J, R1, R2와 5행 5열의 2차원 배열 A를 선언한다.

❷ 5행 5열의 중간 행은 3이므로, 행 위치가 저장될 변수 C를 3으로 초기화한다.
2씩 증가되는 숫자가 저장될 변수 N은 처음 ❼번을 수행한 후 1이 되어야 하므로 –1로 초기화한다.
배열의 행과 열의 위치를 지정할 변수 S와 J를 0으로 초기화한다.

❸ 첫 번째 행에서 열의 시작 위치는 3열이므로, R1을 3(C)으로 초기화한다.

❹ 첫 번째 행에서 열의 끝 위치는 3이므로 R2를 3(C)으로 초기화한다.

❺ 배열의 행은 1부터 5까지 변하므로 행을 지정하는 변수 S를 1씩 증가시킨다.

❻ 배열의 열을 지정하는 변수 J에 열의 시작 위치를 지정하는 변수 R1의 값을 저장한다.

❼ N을 1부터 25까지 2씩 증가시키기 위해 2를 더한다.

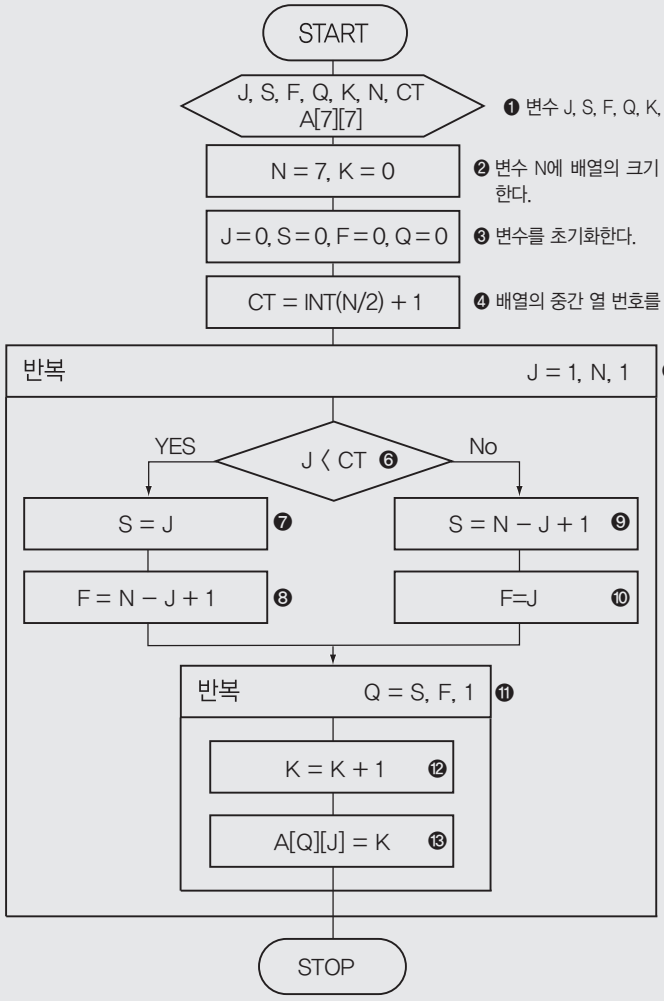
❽ A 배열의 S행 J열(A[S][J])에 N을 저장한다.

[문제 3] Section 041

① J ② S = N – J + 1 ③ F ④ Q ⑤ K = K + 1

변수설명

- A[7][7] : 숫자가 저장될 7행 7열의 2차원 배열
- N : 배열의 크기가 저장될 변수
- K : 1씩 증가되는 숫자가 저장될 변수, 즉 K는 1, 2, 3, …, 31까지 차례로 변경된다.
- Q : 배열의 행 위치를 지정해 주는 변수
- J : 배열의 열 위치를 지정해 주는 변수
- S : 행의 시작 위치를 지정해 주는 변수
- F : 행의 끝 위치를 지정해 주는 변수
- CT : 배열의 중간 열 번호가 저장될 변수



❶ 변수 J, S, F, Q, K, N, CT와 7행 7열의 2차원 배열 A를 선언한다.

❷ 변수 N에 배열의 크기 7을 저장한다. K는 1씩 누적되며 증가하는 변수이므로 0으로 초기화한다.

❸ 변수를 초기화한다.

❹ 배열의 중간 열 번호를 계산한다. 7열이므로 중간 열 번호는 4이다.

❺ 바깥쪽 반복문은 열을 지정한다. 열은 1에서 N(7)까지 변하므로 열을 나타내는 반복 변수 J가 1에서 N(7)이 될 때까지 1씩 증가하면서 ❻~⓭번을 반복하여 수행한 후 끝낸다.

❻ J가 CT(가운데 열)보다 작으면 행의 시작 위치가 J가 되어야 하므로 ❼번으로 가고, 아니면 행의 시작 위치가 (N–J)+1이 되어야 하므로 ❾번으로 간다.

❼ 1열에서 중간 열(CT) 전까지 사용할 행의 시작 위치를 지정한다. 행의 시작 위치는 열 번호 J와 같다.

❽ 1열에서 중간 열(CT) 전까지 사용할 행의 끝 위치를 지정한다. 행의 끝 위치는 열이 증가할 때마다 전체 행수에서 열의 증가한 열 수만큼 줄어든 값에 1을 더한 값만큼 변화한다. 즉 행의 끝 위치는 N–J+1이 된다.

❾ 중간 열(CT) 이후부터 사용할 행의 시작 위치를 지정한다. 행의 시작 위치는 N–J+1이다.

❿ 중간 열(CT) 이후부터 사용할 행의 끝 위치를 지정한다. 행의 끝 위치는 열 번호 J와 같다.

⓫ 안쪽 반복문은 행을 지정한다. 행은 행의 시작 위치 S에서 행의 끝 위치인 F까지 변하므로, 행을 나타내는 반복 변수 Q가 S에서 F가 될 때까지 1씩 증가하면서 ⓬~⓭번을 반복하여 수행한다.

⓬ K를 1에서 31까지 증가시키기 위해 1을 더한다.

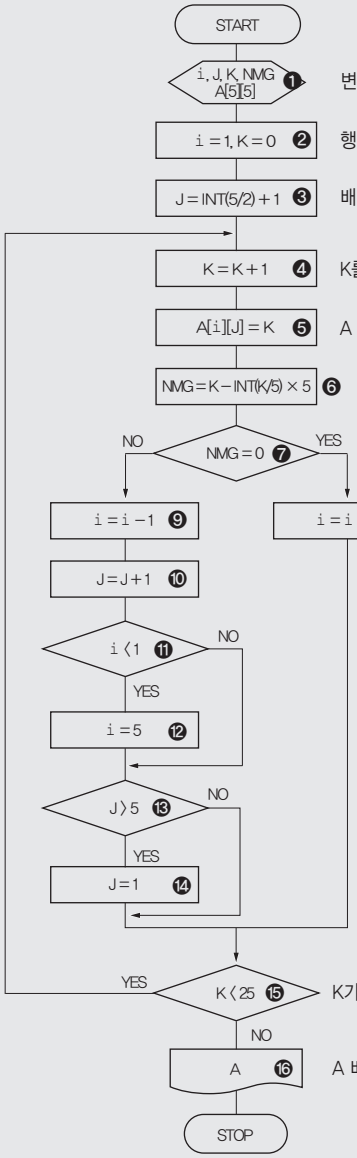
⓭ A 배열의 Q행 J열(A[Q][J])에 K를 저장한다.

[문제 4] Section 046

① YES ② i + 1 ③ J + 1 ④ 5 ⑤ J 〉 5

변수설명

- A[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- i : 배열의 행 위치를 지정해 주는 변수
- J : 배열의 열 위치를 지정해 주는 변수
- K : 1씩 증가되는 숫자가 저장될 변수
- NMG : K가 5의 배수인지를 확인하기 위해 나머지를 구한 후 나머지가 저장될 변수



❶ 변수 i, J, K, NMG와 5행 5열의 2차원 배열 A를 선언한다.

❷ 행 시작 위치는 1이므로 i를 1로 초기화하고, 누적 변수 K는 0으로 초기화한다.

❸ 배열의 가운데 열 번호를 계산한다.

❹ K를 1에서 25까지 증가시키기 위해 1을 더한다.

❺ A 배열의 i행 J열(A[i][J])에 K를 저장한다.

❻ 저장된 숫자가 5의 배수인지를 확인하기 위해 나머지를 계산한다.

❼ NMG가 0이면 5의 배수이므로 행만 1 증가시키기 위해 ❽번으로 가고, 아니면 ❾번으로간다.

❽ 다음 행을 지정하기 위해 행을 1씩 증가시킨다.

❾ 다음 행을 지정하기 위해 행을 1씩 감소시킨다.

❿ 다음 열을 지정하기 위해 열을 1씩 증가시킨다.

⓫ 행을 나타내는 i가 1보다 작으면 행을 5로 지정해야 하므로 ⓬번으로 가고, 아니면 ⓭번으로 간다.

⓬ 다음 행을 5로 만든다.

⓭ 열이 5보다 크면 열을 1로 지정해야 하므로 ⓮번으로 가고, 아니면 ⓯번으로 간다.

⓮ 다음 열을 1로 만든다.

⓯ K가 25보다 작으면 다음 숫자를 채우기 위해 ❹번으로가고, 아니면 ⓰번으로간다.

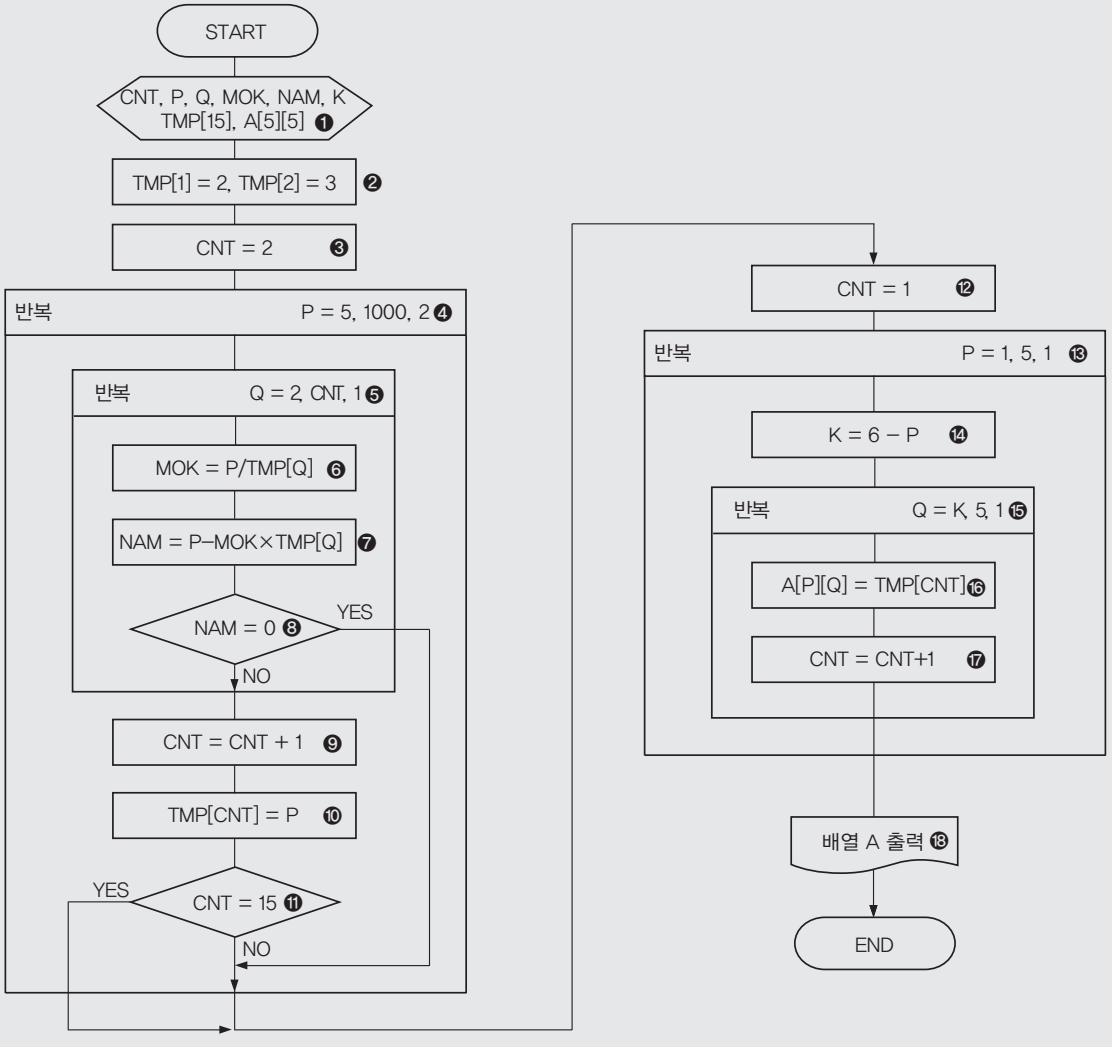
⓰ A 배열을 출력하고 끝낸다.

[문제 5] Section 016, 039

① 2 ② P–MOK×TMP[Q] ③ P ④ 1 ⑤ 6–P

변수설명

- TMP[15] : 소수가 저장될 배열
- A[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- P : 왼쪽 순서도에서는 소수인지 판별할 숫자가 저장될 변수로, 오른쪽 순서도에서는 배열의 행의 위치를 지정해 주는 변수로 사용된다.
- K : 배열의 열의 시작 위치를 지정해 주는 변수
- Q : 배열의 열의 위치를 지정해 주는 변수
- CNT : 1씩 증가되는 숫자가 저장될 변수, 즉 CNT는 1, 2, 3, …, 15까지 차례로 변경된다.
- MOK : 소수 여부를 판별할 때 몫이 저장될 변수
- NAM : 소수 여부를 판별할 때 나머지가 저장될 변수



❶ 사용할 변수와 소수가 저장될 배열, 숫자가 저장될 2차원 배열을 선언한다.

❷ TMP[1]과 TMP[2]에 각각 2와 3을 저장하고 시작한다. 즉 소수 15개 중 첫 번째와 두 번째 소수는 구한 상태로 시작한다.

❸ CNT는 소수의 개수가 누적될 변수인데 첫 번째, 두 번째 소수를 구하고 시작하는 것이므로 2를 저장한다.

❹ 2를 제외한 짝수는 소수가 아니므로 홀수만을 대상으로 하기 위해 반복 변수 P가 5에서 1000이 될 때까지 2씩 증가하면서 ❺~⓫번을 반복 수행한다.

❺ 안쪽 반복문은 대상 숫자가 소수인지를 판별하는 것으로, 반복 변수 Q를 2부터 CNT까지 1씩 증가하면서 ❻~❽번을 반복 수행한다. Q를 2부터 시작하는 이유는 TMP 배열의 첫 번째에 2가 저장되어 있기 때문이다. 소수인지 판별할 수 P가 5, 7, 9, 11...로 증가하는 홀수이기 때문에 2로 나눠서 나머지가 0인지 확인할 필요는 없다.

❻ 몫을 구한다.

❼ 나머지를 구한다.

❽ 나머지가 0이면 소수가 아니므로 다음 수를 판별을 하기 위해 ❹번으로 가고 아니면 소수일 가능성이 있으므로 ❺번으로 간다.

❾ CNT를 1씩 증가시킨다.

❿ TMP 배열에 소수 P를 저장한다.

⓫ CNT가 150이면 소수 15개를 모두 찾은 것이므로 ⓬번으로 가고, 아니면 다음 수를 판별하기 위해 ❹번으로 간다.

⓬ 첫 번째 소수부터 배열에 저장해야 하므로 CNT에 1을 저장한다.

⓭ 반복 변수 P를 1부터 5까지 1씩 증가시키면서 ⓮~⓱번을 반복하여 수행한다.

⓮ 열의 시작 위치를 6-P로 지정한다. 즉 1행에서는 5열부터 시작한다.

⓯ 반복 변수 Q를 K부터 5까지 1씩 증가시키면서 ⓰~⓱번을 반복하여 수행한다.

⓰ A 배열에 소수를 저장한다.

⓱ CNT를 1씩 증가시킨다.

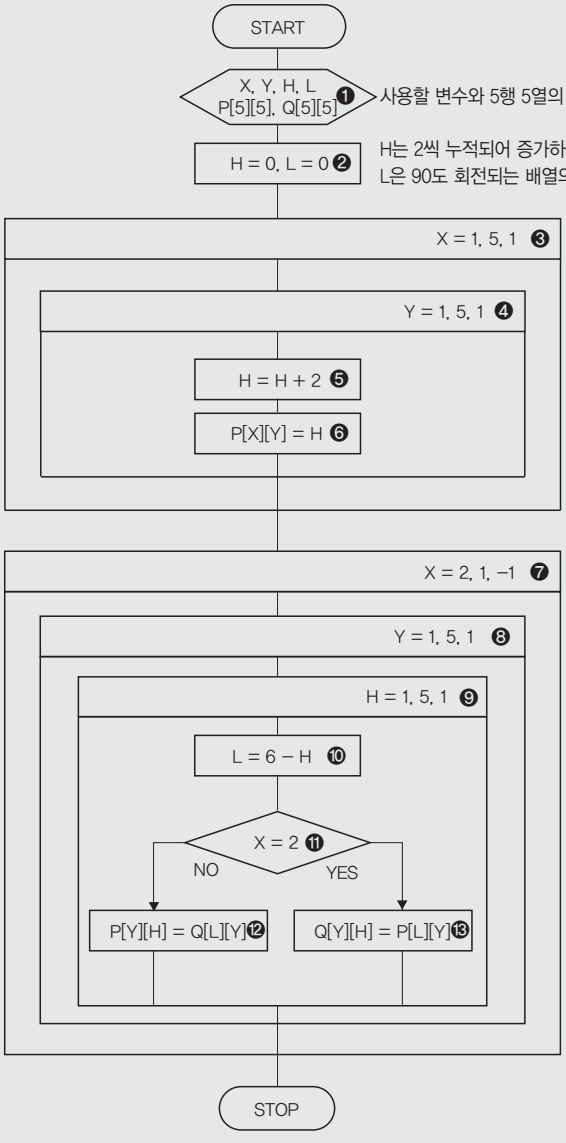
⓲ 배열 A를 출력하고 끝낸다.

[문제 6] Section 045

① H = H + 2 ② Y = 1, 5, 1 ③ L = 6 − H ④ P[Y][H] ⑤ Q[Y][H]

변수설명

- P[5][5] : 처음에는 숫자가 저장되고, 이후 Q 배열의 데이터를 90도 회전하여 이동할 5행 5열의 2차원 배열
- Q[5][5] : P 배열의 데이터를 90도 회전하여 이동할 2차원 배열
- X : 배열의 행 위치를 지정해 주는 변수
- Y : 배열의 열 위치를 지정해 주는 변수
- H : 2씩 증가되는 숫자가 저장될 변수, 즉 H는 2, 4, 6, …, 50까지 차례로 변경된다.
- L : 90도 회전되는 배열의 행 위치를 지정해 주는 변수



※ P 배열에 숫자 저장

❸ 바깥쪽 반복문은 행을 지정한다. 행은 1에서 5까지 변하므로, 행을 나타내는 반복 변수 X가 1에서 5가 될 때까지 1씩 증가하면서 ❹번을 반복하여 수행한다.

❹ 안쪽 반복문은 열을 지정한다. 열을 나타내는 반복 변수 Y가 1에서 5가 될 때까지 1씩 증가하면서 ❺~❻번을 반복하여 수행한다.

❺ H를 2부터 50까지 2씩 증가시키기 위해 2를 더한다.

❻ P 배열에 값을 저장한다.

※ P 배열을 Q 배열로 이동한 후 Q 배열을 P 배열로 다시 이동

❼ 한 번은 P 배열을 Q 배열로 이동하고, 또 한 번은 Q 배열을 P 배열로 이동하는 작업을 수행하기 위해 반복 변수 X를 2에서 1이 될 때까지 −1씩 증가하면서 ❽번을 반복하여 수행한다.

❽ 행을 지정하는 반복문이다. 행을 나타내는 반복 변수 Y가 1에서 5가 될 때까지 1씩 증가하면서 ❾번을 반복하여 수행한다.

❾ 열을 지정하는 반복문이다. 열을 나타내는 반복 변수 H가 1에서 5가 될 때까지 1씩 증가하면서 ❿~⓭번을 반복하여 수행한다.

❿ L은 이동할 배열의 행 위치를 지정하는 변수이다. 이동할 배열의 행은 6−H로 지정한다.

⓫ P 배열을 Q 배열로 이동할지 Q 배열을 P 배열로 이동할지를 판단한다. X가 2일때는 P 배열의 값을 Q 배열로 치환하기 위해 ⓭번으로 가고, 아니면 ⓬으로 간다.

⓬ P 배열에 Q 배열의 값을 치환한다.

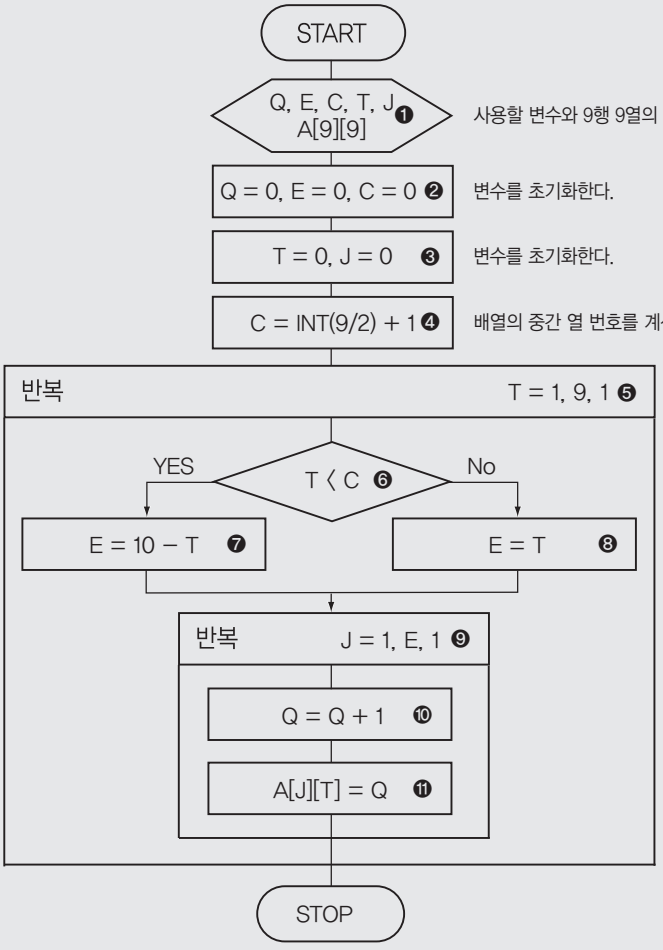
⓭ Q 배열에 P 배열의 값을 치환한다.

[문제 7] Section 043, 044

① C ② E = 10 − T ③ E = T ④ J ⑤ Q = Q + 1

변수설명

- A[9][9] : 숫자가 저장될 9행 9열의 2차원 배열
- Q : 1씩 증가되는 숫자가 저장될 변수
- E : 행의 끝 위치를 지정해 주는 변수
- C : 배열의 중간 열 번호가 저장될 변수, 즉 9행 9열의 배열인 경우 5가 저장된다.
- T : 배열의 열 위치를 지정해 주는 변수
- J : 배열의 행 위치를 지정해 주는 변수



사용할 변수와 9행 9열의 2차원 배열 A를 선언한다.

변수를 초기화한다.

변수를 초기화한다.

배열의 중간 열 번호를 계산한다. 9열이므로 중간 열 번호는 5이다.

❺ 바깥쪽 반복문은 열을 지정한다. 열은 1에서 9까지 변하므로 열을 나타내는 반복 변수 T가 1에서 9가 될 때까지 1씩 증가하면서 ❻~⓫번을 반복하여 수행한 후 끝낸다.

❻ T가 C(가운데 열)보다 작으면 행의 끝 위치가 1씩 감소해야 하므로 ❼번으로 가고, 아니면 행의 끝 위치가 현재 열의 위치와 같아야 하므로 ❽번으로 간다.

❼ 행의 끝 위치를 계산한다. T(현재 열)가 C(가운데 열)보다 작은 경우로 10에서 T를 뺀 값을 행의 끝 위치로 사용한다.

❽ 행의 끝 위치를 계산한다. T(현재 열)가 C(가운데 열)보다 작지 않은 경우로 T값을 행의 끝 위치로 사용한다.

❾ 안쪽 반복문은 행을 지정한다. 행은 1부터 행의 끝 위치인 E까지 변하므로, 행을 나타내는 반복 변수 J가 1에서 E가 될 때까지 1씩 증가하면서 ❿~⓫번을 반복하여 수행한다.

❿ Q를 1에서 65까지 증가시키기 위해 1을 더한다.

⓫ A 배열의 J행 T열(A[J][T])에 Q를 저장한다.

[문제 8] Section 040

① break ② j = –1

변수설명

- a[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- k : 1씩 증가되는 숫자가 저장될 변수, 즉 k는 1, 2, 3, …, 25까지 차례로 변경된다.
- **i** : 배열의 행 위치를 지정해 주는 변수
- **j** : 배열의 열 위치를 지정해 주는 변수
- n : 열 위치의 증가 혹은 감소 여부를 지정해 주는 변수

```
#include <stdio.h>

main()
{
❶ int k, i, j, n;
  int a[5][5] = { 0 };
❷ k = 0, i = j = -1;          배열의 위치가 0부터 시작하기 때문에 i와 j는 ❺번과 ❿번을 수행한 후 0이 되도록 모두 –1로 초기화한다.
❸ n = -1;                     첫 행에서 열의 증가 값은 1이지만 ❻번을 실행한 후 1이 되도록 –1로 초기화한다.
❹ while(1)                    while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❺~㉔번 문장을 무한 반복한다.
  {
❺   i++;                      i는 행을 지정한다. 행을 지정하는 i를 0에서 4까지 증가시키기 위해 1을 더한다.
❻   n *= -1;                  열 위치의 증가값을 1 또는 –1로 변경하기 위해 n에 –1을 곱한다
❼   if (i > 4)                행의 위치가 0~4까지 변하기 때문에 i가 4보다 커져 모든 수행이 끝났는지 확인한다.
❽     break;                  while 반복문을 탈출한다. 제어가 ㉓번으로 이동한다.
❾   while (1)                 while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❿~㉔번 문장을 무한 반복한다.
    {
❿     j += n;                 j는 열을 지정한다. 열을 지정하는 j를 0에서 4 또는 4에서 0까지 증가시키기 위해 j에 n을 누적한다.
⓫     k++;                    k는 1에서 25까지 증가시키기 위해 1을 더한다.
⓬     a[i][j] = k;            a 배열에 i행 j열(a[i][j])에 k를 저장한다.
⓭     if (i % 2 == 0)         배열의 위치가 0부터 시작하기 때문에 i가 짝수이면 ⓮번으로 가고, 아니면 ⓴번으로 간다.
      {
⓮       if (j < 4)            이곳으로 온 경우는 짝수 행으로 열의 위치가 0~4까지 변하기 때문에 열이 4보다 작으면 ⓯번으로 가고, 아니면 ⓱번으로 간다.
⓯         continue;           제어가 while문의 시작점으로 이동하여 ❿번부터 다시 시작한다.
⓰       else
        {
⓱         j++;                열을 나타내는 j가 4가 되었을 경우이다. 다음 행의 열 j는 4~0까지 변해야 하므로 5를 만들기 위해 j에 1을 더한다.
⓲         break;              while 반복문을 탈출한다. 제어가 ㉓번으로 이동한다.
        }
      }
⓳     else
      {
⓴       if (j > 0)            이곳으로 온 경우는 홀수 행으로 열의 위치가 4~0까지 변화하기 때문에 열이 0보다 크면 ㉑번으로 가고, 아니면 ㉔번으로 간다.
㉑         continue;           제어가 while문의 시작점으로 이동하여 ❿번부터 다시 시작한다.
㉒       else
        {
㉓         j = -1;             열을 나타내는 j가 0이 되었을 경우이다. 다음 행의 열 j는 0~4까지 변해야 하므로 j를 –1로 초기화한다.
㉔         break;              while 반복문을 탈출한다. 제어가 ㉓번으로 이동한다.
        }
```

```
        }
      }
㉕ }
㉖ for (int x = 0; x <= 4; x++)          a 배열의 내용을 출력하고 끝낸다.
  {
    for (int y = 0; y <= 4; y++)
      printf("%3d", a[x][y]);
    printf("\n");
  }
}
```

[문제 9] Section 042

① 2　② s < 4

변수설명

- c : 행의 중간 위치가 저장될 변수
- n : 2씩 증가되는 숫자가 저장될 변수, 즉 n은 1, 3, 5, …, 23, 25까지 차례로 변경된다.
- s : 배열의 행 위치를 지정해 주는 변수
- j : 배열의 열 위치를 지정해 주는 변수
- r1 : 열의 시작 위치가 저장될 변수, 즉 r1이 2이면 2열부터 데이터를 저장한다.
- r2 : 열의 끝 위치가 저장될 변수, 즉 r2가 4이면 4열까지 데이터를 저장한다.

```
#include <stdio.h>

main()
{
❶ int c, n, s, j, r1, r2;
  int a[5][5] = { 0 };

❷ c = 2, n = s = -1;          배열의 위치가 0부터 시작하기 때문에 5행 5열의 중간 행은 2이므로, 행 위치가 저장될 변수 c를 2로 초기화한다.
                              2씩 증가되는 숫자가 저장될 변수 n은 ❾번을 수행한 후 1이 되도록 -1로 초기화한다.
                              배열의 위치가 0부터 시작하기 때문에 배열의 행 위치를 지정할 변수 s는 ❻번을 수행한 후 0이 되도록 -1로 초기화한다.
❸ r1 = c;                     첫 번째 행에서 열의 시작 위치는 2열이므로, r1을 2(c)로 초기화한다.
❹ r2 = c;
❺ do                          do~while 반복문의 시작점으로 ❻~⓲번 문장을 반복한다.
  {
❻   s++;                     배열의 행은 1부터 5까지 변하므로 행을 지정하는 변수 s를 1씩 증가시킨다.
❼   j = r1;                  배열의 열을 지정하는 변수 j에 열의 시작 위치를 지정하는 변수 r1의 값을 저장한다
❽   do                       do~while 반복문의 시작점으로 ❾~⓫번 문장을 반복한다.
    {
❾     n += 2;                    n을 1부터 25까지 2씩 증가시키기 위해 2를 더한다.
❿     a[s][j] = n;               a 배열의 s행 j열(a[s][j])에 n을 저장한다.
⓫     j++;                       열의 위치를 1씩 증가시킨다.
⓬   } while (j <= r2);           열의 위치는 열의 시작 위치인 r1부터 열의 끝 위치인 r2까지 변하므로 j가 r2보다 작거나 같으면 현재 행을 다 채우지
                                 않았으므로 ❾번으로 가고, 아니면 다음 행을 채우기 위해 ⓭번으로 간다.
⓭   if (s >= c)              현재 작업한 행(s)이 2(c)보다 크거나 같으면 열의 시작 위치 r1은 1씩 증가시키고, 열의 끝 위치 r2는 -1씩 증가시키기 위해
                              ⓮번으로 가고, 아니면 반대의 작업을 수행하기 위해 ⓱번으로 간다.
    {
⓮     r1++;                      열의 시작 위치 r1을 1씩 증가시킨다.
⓯     r2--;                      열의 끝 위치 r2를 -1씩 증가시킨다.
    }
⓰   else
    {
⓱     r1--;                    열의 시작 위치 r1을 -1씩 증가시킨다.
⓲     r2++;                    열의 끝 위치 r2를 1씩 증가시킨다.
    }
⓳ } while (s < 4);               배열의 위치가 0~4까지 변하므로 s가 4보다 작으면 아직 채워야 할 행이 남았으므로 ❻번으로 가고, 아니면 다 채운 것이므로
                                 끝낸다.
}
```

[문제 10] Section 041

① n / 2　② j < ct

변수설명

- a[7][7] : 숫자가 저장될 7행 7열의 2차원 배열
- n : 배열의 크기가 저장될 변수
- k : 1씩 증가되는 숫자가 저장될 변수, 즉 k는 1, 2, 3, …, 31까지 차례로 변경된다.
- q : 배열의 행 위치를 지정해 주는 변수
- j : 배열의 열 위치를 지정해 주는 변수
- s : 행의 시작 위치를 지정해 주는 변수
- f : 행의 끝 위치를 지정해 주는 변수
- ct : 배열의 중간 열 번호가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int j, s, f, q, k, n, ct;
  int a[7][7] = { 0 };

❷ n = 7, k = 0;
❸ j = s = f = q = -1;          배열의 위치가 0부터 시작하기 때문에 행과 열의 위치를 지정해 주는 변수들을 모두 -1로 초기화한다.
❹ ct = n / 2;                  C언어에서 정수형 변수는 소수점이하는 버리고 정수만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다. 배열의 위치가 0부터 시작하기 때문에 0에서 6까지 증가하는 배열의 중간 열 번호는 2로만 나누면 된다.
❺ for (j = 0; j <= n - 1; j++)   배열의 위치가 0부터 시작하기 때문에 열 첨자 j는 0부터 (n-1)까지 변하는 동안 ❻~⓭번을 반복 수행한다.
  {
❻   if (j < ct)                 j가 ct(가운데 열)보다 작으면 ❼번으로 가고, 아니면 ❾번으로 간다.
    {
❼     s = j;                   1열에서 중간 열(ct) 전까지 사용할 행의 시작 위치다. 행의 시작 위치는 열 번호 j와 같다.
❽     f = n - j;               행의 끝 위치는 열이 증가할 때마다 전체 행수에서 증가한 열의 수만큼 줄어든 값에 1을 더한 값만큼 변화해야 하지만 배열의 위치가 0부터 시작하기 때문에 행의 끝 위치는 n-j가 된다.

    }
    else
    {
❾     s = n - (j + 1);         중간 열 이후의 행 시작 위치는 열이 증가할 때마다 전체 행수에서 증가한 열의 수만큼 줄어든 값에 1을 더한 값만큼 변화해야 하지만 배열의 위치가 0부터 시작하기 때문에 전체 행수에서 증가한 열의 수에 1을 더한 수만큼 줄어들어야 한다.
❿     f = j + 1;               중간 열 이후의 행 끝 위치는 열 번호 j와 같아야 하지만 배열의 위치가 0부터 시작하기 때문에 j에 1을 더한 수가 되어야 한다.

    }
⓫   for (q = s; q <= f - 1; q++)   안쪽 반복문은 행을 지정한다. 행은 행의 시작 위치인 s에서 행의 끝 위치인 f여야 하지만 배열의 위치가 0부터 시작하기 때문에 f-1까지 반복하여 수행해야 한다.
    {
⓬     k++;                     k를 1에서 31까지 증가시키기 위해 1을 더한다.
⓭     a[q][j] = k;             a 배열의 a행 j열(a[q][j])에 k를 저장한다.
    }
  }
}
```

[문제 11] Section 046

① nmg == 0　② j = 0

변수설명

- a[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- **i** : 배열의 행 위치를 지정해 주는 변수
- **j** : 배열의 열 위치를 지정해 주는 변수
- k : 1씩 증가되는 숫자가 저장될 변수
- nmg : k가 5의 배수인지를 확인하기 위해 나머지를 구한 후 나머지가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int i, j, k, nmg;
  int a[5][5] = { 0 };

❷ i = 0, k = 0;                  배열의 위치가 0부터 시작하기 때문에 행의 시작 위치를 지정해 주는 i를 0으로 초기화한다.
❸ j = 5 / 2;                     C언어에서 정수형 변수는 소수점이하는 버리고 정수만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다. 배열의 위치가
                                 0부터 시작하기 때문에 0에서 4까지 증가하는 배열의 중간 열 번호는 2로만 나누면 된다.
  do                             do~while 반복문의 시작점으로 ❹~⓮번 문장을 반복한다.
  {
❹   k++;                         k를 1에서 25까지 증가시키기 위해 1을 더한다.
❺   a[i][j] = k;                 a 배열의 i행 j열(a[i][j])에 k를 저장한다.
❻   nmg = k - k / 5 * 5;         저장된 숫자가 5의 배수인지를 확인하기 위해 나머지를 계산한다.
❼   if (nmg == 0)                nmg가 0이면 5의 배수이므로 ❽번으로 가고, 아니면 ❾번으로 간다.
❽     i++;                       다음 행을 지정하기 위해 행을 1씩 증가시킨다.
    else
    {
❾     i--;                       다음 행을 지정하기 위해 행을 1씩 감소시킨다.
❿     j++;                       다음 열을 지정하기 위해 열을 1씩 증가시킨다.
⓫     if (i < 0)                 행의 위치가 0~4까지 변하기 때문에 행이 0보다 작아져 배열의 크기를 벗어나면 ⓬번으로 가고, 아니면 ⓭번으로 간다.
⓬       i = 4;                   행이 0보다 작아진 경우 행의 위치를 마지막인 4로 변경한다.
⓭     if (j > 4)                 열의 위치가 0~4까지 변화하기 때문에 열이 4보다 커져 배열의 크기를 벗어나면 ⓮번으로 가고, 아니면 ⓯번으로 간다.
⓮       j = 0;                   열이 4보다 커진 경우 열의 위치를 처음인 0으로 변경한다.
    }
⓯ } while (k < 25);              k가 25보다 작으면 다음 숫자를 채우기 위해 ❹번으로 가고, 아니면 ⓰번으로 간다.
⓰ for (int x = 0; x <= 4; x++)       a 배열을 출력하고 끝낸다.
  {
    for (int y = 0; y <= 4; y++)
      printf("%3d", a[x][y]);
    printf("\n");
  }
}
```

[문제 12] Section 016, 039

① tmp[q]　② cnt = 0

변수설명

- tmp[15] : 소수가 저장될 배열
- a[5][5] : 숫자가 저장될 5행 5열의 2차원 배열
- p : 소수인지 판별할 숫자가 저장될 변수와 배열의 행의 위치를 지정해 주는 변수로 사용된다.
- k : 배열의 열의 시작 위치를 지정해 주는 변수
- q : 배열의 열의 위치를 지정해 주는 변수
- cnt : 1씩 증가되는 숫자가 저장될 변수, 즉 cnt는 1, 2, 3, …, 15까지 차례로 변경된다.
- mok : 소수 여부를 판별할 때 몫이 저장될 변수
- nam : 소수 여부를 판별할 때 나머지가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int cnt, p, q, mok, nam, k;
  int tmp[15] = { 0 };
  int a[5][5] = { 0 };

❷ tmp[0] = 2, tmp[1] = 3;          배열의 위치가 0부터 시작하기 때문에 tmp의 첫 번째와 두 번째 요소는 tmp[0]과 tmp[1]이다.
❸ cnt = 1;                         배열의 첨자로 사용하는 cnt는 두 번째 소수를 구하고 시작하는 것이므로 2로 해야 하지만 배열의 위치가 0부터 시작하기
                                   때문에 1로 초기화한다.
❹ for (p = 5; p <= 1000; p += 2)   2를 제외한 짝수는 소수가 아니므로 홀수만을 대상으로 하기 위해 반복 변수 p가 5에서 1000이 될 때까지 2
                                   씩 증가하면서 ❺~⓬번을 반복 수행한다.
  {
❺   for (q = 1; q <= cnt; q++)     안쪽 반복문은 대상 숫자가 소수인지를 판별하는 것으로, 반복 변수 q를 2부터 시작해야 하지만 배열의 위치
                                   가 0부터 시작하기 때문에 1부터 시작해서 cnt가 될 때까지 1씩 증가하면서 ❻~❽번을 반복 수행한다. q를 1
                                   부터 시작하는 이유는 tmp 배열의 첫 번째에 2가 저장되어 있기 때문이다. 소수인지 판별할 수 p가 5, 7, 9,
                                   11…로 증가하는 홀수이기 때문에 2로 나눠서 나머지가 0인지 확인할 필요는 없다.
    {
❻     mok = p / tmp[q];            몫을 구한다.
❼     nam = p - mok*tmp[q];        나머지를 구한다.
❽     if (nam == 0)                나머지가 0이면 소수가 아니므로 다음 수를 판별을 하기 위해 ❾번으로 가고, 아니면 ❺번으로 간다.
        break;                     안쪽 반복문(for)을 탈출한다. 제어가 ❾번으로 간다.
    }
❾   if (nam == 0)                  ❽번 조건을 만족한 경우 break에 의해 안쪽 반복문을 빠져나와 바깥쪽 반복문의 시작 위치로 이동해야 하지만 안쪽 반복
                                   문을 빠져나오면 처리문(⓾~⓬)이 있기 때문에 이 처리문을 수행하지 않고 바로 바깥쪽 반복문의 시작 위치로 이동하기 위
                                   해 한 번 더 ❽번의 조건을 비교한다.
      continue;                    제어가 바깥쪽 반복문(for)의 시작 위치인 ❹로 이동한다.
⓾   cnt++;                         cnt를 1씩 증가시킨다.
⓫   tmp[cnt] = p;                  tmp 배열에 소수 p를 저장한다.
⓬   if (cnt == 14)                 배열의 위치가 0부터 시작하기 때문에 cnt는 0~14까지 15개 소수를 저장하기 위한 위치로 사용된다. cnt
                                   가 14이면 소수 15개를 모두 찾은 것이므로 ⓭번으로 가고, 아니면 바깥쪽 반복문(for)의 시작 위치인 ❹로 이
                                   동한다.
      break;                       바깥쪽 반복문(for)을 탈출한다. 제어가 ⓭번으로 간다.
```

```
    }
⑬  cnt = 0;                          배열의 위치가 0부터 시작하기 때문에 첫 번째 소수의 위치를 가리키는 첨자로 0을 사용한다.
⑭  for (p = 0; p <= 4; p++)          배열의 위치가 0부터 시작하기 때문에 행 첨자 p는 0~4까지 변하는 동안 ⑮~⑱번을 반복 수행한다.
    {
⑮     k = 4 - p;                     행 첨자 p가 0부터 시작하기 때문에 열의 시작 위치는 4-p로 지정한다. 즉 0행에서는 4열부터 시작한다.
⑯     for (q = k; q <= 4; q++)       배열의 위치가 0부터 시작하기 때문에 열 첨자 q는 k에서 시작해서 4까지 변하는 동안 ⑰~⑱번을 반복 수행
                                     한다.
      {
⑰       a[p][q] = tmp[cnt];          a 배열에 소수를 저장한다.
⑱       cnt++;                       cnt를 1씩 증가시킨다.
      }
    }
⑲  for (int x = 0; x <= 4; x++)      배열 a를 출력하고 끝낸다.
    {
      for (int y = 0; y <= 4; y++)
        printf("%3d", a[x][y]);
      printf("\n");
    }
}
```

[문제 13] Section 045

① x == 2 ② p[L][y] ③ q[L][y]

변수설명

- p[5][5] : 처음에는 숫자가 저장되고, 이후 q 배열의 데이터를 90도 회전하여 이동할 5행 5열의 2차원 배열
- q[5][5] : p 배열의 데이터를 90도 회전하여 이동할 2차원 배열
- x : 배열의 행 위치를 지정해 주는 변수
- y : 배열의 열 위치를 지정해 주는 변수
- h : 2씩 증가되는 숫자가 저장될 변수, 즉 h는 2, 4, 6, …, 50까지 차례로 변경된다.
- L : 90도 회전되는 배열의 행 위치를 지정해 주는 변수

```
#include <stdio.h>

main()
{
❶ int x, y, h, L;
  int p[5][5] = { 0 }, q[5][5] = { 0 };

❷ h = L = 0;
❸ for (x = 0; x <= 4; x++)                  배열의 위치가 0부터 시작하기 때문에 행 첨자 x는 0~4까지 변하는 동안 ❹번을 반복 수행한다.
  {
❹   for (y = 0; y <= 4; y++)                배열의 위치가 0부터 시작하기 때문에 열 첨자 y는 0~4까지 변하는 동안 ❺~❻번을 반복 수행한다.
    {
❺     h += 2;                               h를 2부터 50까지 2씩 증가시키기 위해 2를 더한다.
❻     p[x][y] = h;                          p 배열에 값을 저장한다.
    }
  }
❼ for (x = 2; x >= 1; x--)                  한 번은 p 배열을 q 배열로 이동하고, 또 한 번은 q 배열을 p 배열로 이동하는 작업을 수행하기 위해 반복 변수
                                            x를 2에서 1이 될 때까지 –1씩 증가하면서 ❽번을 반복 수행한다.
  {
❽   for (y = 0; y <= 4; y++)                배열의 위치가 0부터 시작하기 때문에 행 첨자 y는 0~4까지 변하는 동안 ❾번을 반복 수행한다.
    {
❾     for (h = 0; h <= 4; h++)              배열의 위치가 0부터 시작하기 때문에 열 첨자 h는 0~4까지 변하는 동안 ❿~⓭번을 반복 수행한다.
      {
❿       L = 4 - h;                          열 첨자 h가 0부터 시작하기 때문에 이동할 배열의 행은 4–h로 지정한다. 즉 h가 0~4까지 변하는 동안 L은 4~0까지 변
                                            한다.
⓫       if (x == 2)                         p 배열을 q 배열로 이동할 지 q 배열을 p 배열로 이동할 지를 판단한다. x가 2일때는 p 배열의 값을 q 배열로 치환하기 위
                                            해 ⓬번으로 가고, 아니면 ⓭번으로 간다.
⓬         q[y][h] = p[L][y];                p 배열에 q 배열의 값을 치환한다.
        else
⓭         p[y][h] = q[L][y];                q 배열에 p 배열의 값을 치환한다.
      }
    }
  }
}
```

[문제 14] Section 043, 044

① t < c　② a[j][t]

변수설명

- a[9][9] : 숫자가 저장될 9행 9열의 2차원 배열
- q : 1씩 증가되는 숫자가 저장될 변수
- e : 행의 끝 위치를 지정해 주는 변수
- c : 배열의 중간 열 번호가 저장될 변수, 즉 9행 9열의 배열인 경우 5가 저장된다.
- t : 배열의 열 위치를 지정해 주는 변수
- j : 배열의 행 위치를 지정해 주는 변수

```
#include <stdio.h>

main()
{
❶ int q, e, c, t, j;
  int a[9][9] = { 0 };

❷ q = e = c = 0;
❸ t = j = 0;
❹ c = 9 / 2;
```

C언어에서 정수형 변수는 소수점이하는 버리고 정수만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다. 배열의 위치가 0부터 시작하기 때문에 0에서 4까지 변하는 배열의 중간 열 번호는 2로만 나누면 된다.

```
❺ for (t = 0; t <= 8; t++)
  {
```

배열의 위치가 0부터 시작하기 때문에 열 첨자 t는 0~8까지 변하는 동안 ❻~⓫번을 반복 수행한다.

```
❻   if (t < c)
```

t가 c(가운데 열)보다 작으면 행의 끝 위치가 1씩 감소해야 하므로 ❼번으로 가고, 아니면 행의 끝 위치가 현재 열의 위치와 같아야 하므로 ❽번으로 간다.

```
❼     e = 8 - t;
    else
❽     e = t;
```

열 첨자 t가 0부터 시작하기 때문에 행위 끝 위치인 e는 8-t로 지정한다. 즉 t가 0~8까지 변하는 동안 e은 8~0까지 변한다.

행의 끝 위치를 계산한다. t(현재 열)가 c(가운데 열)보다 작지 않은 경우로 t값을 행의 끝 위치로 사용한다.

```
❾   for (j = 0; j <= e; j++)
    {
```

배열의 위치가 0부터 시작하기 때문에 행 첨자 j는 0에서 시작해서 e까지 변하는 동안 ❿~⓫번을 반복 수행한다.

```
❿     q++;
⓫     a[j][t] = q;
    }
  }
}
```

q를 1에서 65까지 증가시키기 위해 1을 더한다.

a 배열의 j행 t열(a[j][t])에 q를 저장한다.

Section 050

① 50000 ② INT(T / M) ③ T-(DATA.PM[K]*M) ④ M / 2 ⑤ TM[K]+DATA.PM[K]

※ 아래 디버깅 표는 배열(TM, DATA.PM)을 초기화시키는 과정을 생략한 것입니다.

N	P	T	M	SW	K	배열	출력
엄철식	539620	539620 39620 9620 4620 620 120 20 20 0 0	50000 10000 5000 1000 500 100 50 10 5 1	1 0 1 0 1 0 1 0 1 0	1 2 3 4 5 6 7 8 9	DATA.PM 10 3 1 4 1 1 0 2 0 0 TM 10 3 1 4 1 1 0 2 0 0	출장비 지급표 성명 출장비 오만 만 오천 천 오백 백 오십 십 오 일 엄철식 539620 10 3 1 4 1 1 0 2 0 0
양동수	538973	538973 38973 8973 3973 973 473 73 23 3 3	50000 10000 5000 1000 500 100 50 10 5 1	1 0 1 0 1 0 1 0 1 0	1 2 3 4 5 6 7 8 9	DATA.PM 10 3 1 3 1 4 1 2 0 3 TM 20 6 2 7 2 5 1 4 0 3	출장비 지급표 성명 출장비 오만 만 오천 천 오백 백 오십 십 오 일 엄철식 539620 10 3 1 4 1 1 0 2 0 0 양동수 538973 10 3 1 3 1 4 1 2 0 3
이동훈	173105	173105 23105 3105 3105 105 105 5 5 5 0	50000 10000 5000 1000 500 100 50 10 5 1	1 0 1 0 1 0 1 0 1 0	1 2 3 4 5 6 7 8 9	DATA.PM 3 2 0 3 0 1 0 0 1 0 TM 23 8 2 10 2 6 1 4 1 3	출장비 지급표 성명 출장비 오만 만 오천 천 오백 백 오십 십 오 일 엄철식 539620 10 3 1 4 1 1 0 2 0 0 양동수 538973 10 3 1 3 1 4 1 2 0 3 이동훈 173105 3 2 0 3 0 1 0 0 1 0
QUIT							출장비 지급표 성명 출장비 오만 만 오천 천 오백 백 오십 십 오 일 엄철식 539620 10 3 1 4 1 1 0 2 0 0 양동수 538973 10 3 1 3 1 4 1 2 0 3 이동훈 173105 3 2 0 3 0 1 0 0 1 0 전체 회폐 매수 23 8 2 10 2 6 1 4 1 3

Section 051

① BUTOT = 0　② YES　③ NO　④ BUBI = DATA.BU

DATA.BU	DATA.IRUM	DATA.BON	DATA.SU	BUTOT	BUBI	KEB	GTOT	출력
영업	강현준	1000	100	0	영업	1100	0	사원 급여표
영업	조충희	1300	140	1100	총무	1440	2540	부서 성명 본봉 수당 합계
총무	이다인	1200	150	2540		1350	5620	영업 강현준 1000 100 1100
총무	강호정	1500	230	0		1730		영업 조충희 1300 140 1440
				1350				부서 합계 : 2540
				3080				총무 이다인 1200 150 1350
								총무 강호정 1500 230 1730
								부서합계 : 3080
								전체합계 : 5620

Section 052

① DATA.NAI >= 60　② ROW = DATA.DONG　③ COL = INT(DATA.NAI / 10) + 1

DATA.DONG	DATA.IRUM	DATA.NAI	ROW	COL
3	김말동	35	3	4
1	유진호	72	1	7
3	김유라	60	3	4
0		30		

배열 A

0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0
0	0	0	2	0	0	0	2
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	2	0	0	1	3

Section 053

① i < K - 1 ② FLAG = 0 ③ C = DATA[i].BAN

FLAG	BAN	BUNHO	WGT	A	B	C	출력		
0	1반	10	40	40	40	1반	반	번호	체중
1	1반	11	30	30	48	2반	1반	10	40
	2반	4	35					11	30
	2반	8	48				2반	4	35
								8	48
							최소 체중 : 30		
							최대 체중 : 48		

Section 054

① CNT = 0 ② CNT = CNT + 1

i	J	K	HAP	CA=NT	출력
1	1 2 3	1 2 3 1 2 3 1 2 3	900 1300 1700 1200 1600 2000 1500 1900 2300	0 1	1, 1, 2, 3
2	1 2 3	1 2 3 1 2 3 1 2 3	1100 1500 1900 1400 1800 2200 1700 2100 2500		
3	1 2 3	1 2 3 1 2 3 1 2 3	1300 1700 2100 1600 2000 2400 1900 2300 2700	2	2, 3, 2, 2

Section **055**

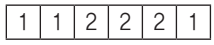
① J = i　② NMG = 1　③ J = J × 3 / 4

i	J	K	NMG	출력
1021	1021	1	1	1021
	1020	2	1	
	765	3	1	
	764	4	1	
	573	5	1	
	572			
	429			
	428			
	321			
	320			
	240			

Section **056**

① BAN = 1 ② A[NMG] = BAN ③ NMG = 1 ④ i = i – 1

X1	X2	HAP	i	NMG	CNT	BAN	배열 A
2	6	8	1	2	0	0	
			2	3	1	1	
			1	4	2	2	
			2	5	3	3	
			1	6	4	1	
			2	7	5	2	
			1	1	6	3	
			2	2		1	
			1	3		2	
			0	4			
			1	5			
			2	6			
			1	7			
			2	1			
			1	2			
			0	3			
			1	4			
			0	5			
			1	6			
			0	7			
			1	1			
			2	2			
			1	3			
			2	4			
			1	5			
			2				
			1				
			2				
			1				
			2				
			1				
			2				



Section 057

① K = X ② X = MOK ③ A[i] = 0 ④ M = M + A[6]

X	K	i	MOK	NMG	J	M	배열 A	출력
123456 12345 1234 123 12 1 0 777777	123456	1 2 3 4 5 6 1	12345 1234 123 12 1 0	6 5 4 3 2 1	1 2 3 4 5	0 6 60 65 650 654 6540 6543 65430 65432 654320 654321	6 5 4 3 2 1	123456, 654321, 777777

Section 058

① J = i − 1 ② i = i + 1 ③ J = J − 1

i	J	M	K	A[i]	A[J]	배열 A
1 2 3 4 5 6 1 2 3	5 4 3	3	6 7 8	4 2 8	6 7 8	6 7 8 2 4 0 4 2 8 7 6

Section **059**

① A[C] = B × C ② C 〈 9 ③ B 〈 9

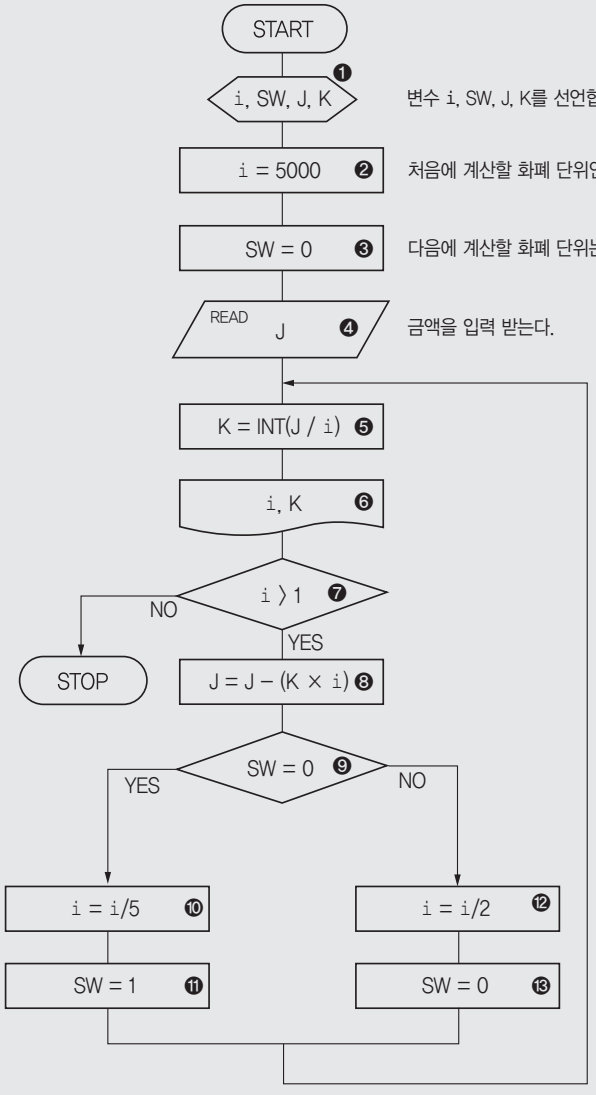
B	C	A[C]	출력
0	0	1	*** 구구단 ***
1	1	2	1 × 1 = 1
	2	3	1 × 2 = 2
	3	4	1 × 3 = 3
	4	5	1 × 4 = 4
	5	6	1 × 5 = 5
	6	7	1 × 6 = 6
	7	8	1 × 7 = 7
	8	9	1 × 8 = 8
	9		1 × 9 = 9
3	0	3	3 × 1 = 3
	1	6	3 × 2 = 6
	2	9	3 × 3 = 9
	3	12	3 × 4 = 12
	4	15	3 × 5 = 15
	5	18	3 × 6 = 18
	6	21	3 × 7 = 21
	7	24	3 × 8 = 24
	8	27	3 × 9 = 27
	9		
⋮	⋮	⋮	⋮
8	0	8	8 × 1 = 8
	1	16	8 × 2 = 16
	2	24	8 × 3 = 24
	3	32	8 × 4 = 32
	4	40	8 × 5 = 40
	5	48	8 × 6 = 48
	6	56	8 × 7 = 56
	7	64	8 × 8 = 64
	8	72	8 × 9 = 72
	9		
9	0	9	9 × 1 = 9
	1	18	9 × 2 = 18
	2	27	9 × 3 = 27
	3	36	9 × 4 = 36
	4	45	9 × 5 = 45
	5	54	9 × 6 = 54
	6	63	9 × 7 = 63
	7	72	9 × 8 = 72
	8	81	9 × 9 = 81
	9		

[문제 1] Section 050

① i 〉 1　② i = i / 5　③ i = i / 2

변수설명

- **i** : 화폐의 단위가 저장될 변수로, 50000, 10000, 5000, 1000, ..., 1까지 차례로 변경된다.
- SW : 스위치 변수
- J : 금액이 저장될 변수
- K : 화폐의 매수가 저장될 변수



변수 i, SW, J, K를 선언합니다.

처음에 계산할 화폐 단위인 50000으로 i를 초기화한다.

다음에 계산할 화폐 단위는 10000으로, 50000을 5로 나누기 위해 SW를 0으로 초기화한다.

금액을 입력 받는다.

❺ 화폐 매수를 계산한다. 입력 받은 금액을 화폐 단위로 나눈 후 정수 값만 K에 저장한다.

❻ 화폐 단위와 화폐 매수를 출력한다.

❼ 화폐 매수 계산의 완료 여부를 판단한다. 화폐 단위가 50000에서 시작하여 1이 되면 매수 계산이 끝난다는 것을 염두에 두고 생각해 보시오.

❽ 화폐 매수를 구한 금액을 뺀 나머지 금액을 계산한다.

❾ SW가 0이면 화폐 단위를 5로 나누기 위해 ❿번으로 가고, 아니면 2로 나누기 위해 ⓬번으로 간다.

❿ 10000, 1000, 100, 10, 1에 대한 화폐 단위를 구한다.

⓫ 다음 번에는 2로 나누기 위해 SW를 1로 치환한다.

⓬ 5000, 500, 50, 5에 대한 화폐 단위를 구한다.

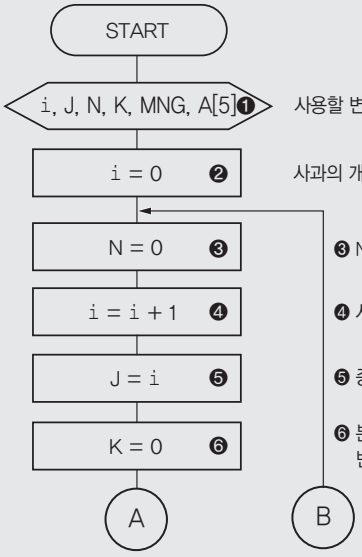
⓭ 다음 번에는 5로 나누기 위해 SW를 0으로 치환한다.

[문제 2] Section 045

① i + 1 ② J – INT(J/4) × 4 ③ J – 1 ④ J × 3/4 ⑤ A[5]

변수설명

- A[5] : 각 사람이 가질 사과의 개수가 저장될 배열
- i : 사과의 개수를 셀 변수, 즉 i는 1부터 조건에 맞는 수가 될 때까지 차례로 변경된다.
- J : 계산에 사용될 사과의 개수가 저장될 변수, 즉 J는 i를 그대로 받아서 계산한다.
- N : 각 사람이 갖는 사과의 개수를 저장시키기 위한 배열의 위치를 지정해 주는 변수
- K : 분배 횟수를 지정해 주는 변수, 4등분하고 1개가 남는 조건을 5번 반복해야 하므로 K는 1에서 5까지 차례로 변경된다.
- NMG : J를 4로 나눈 나머지와 4등분한 사과를 네 사람이 한 몫씩 가진 후 나머지 사과의 개수가 저장될 변수



사용할 변수와 사과의 개수를 저장할 배열 A를 선언한다.

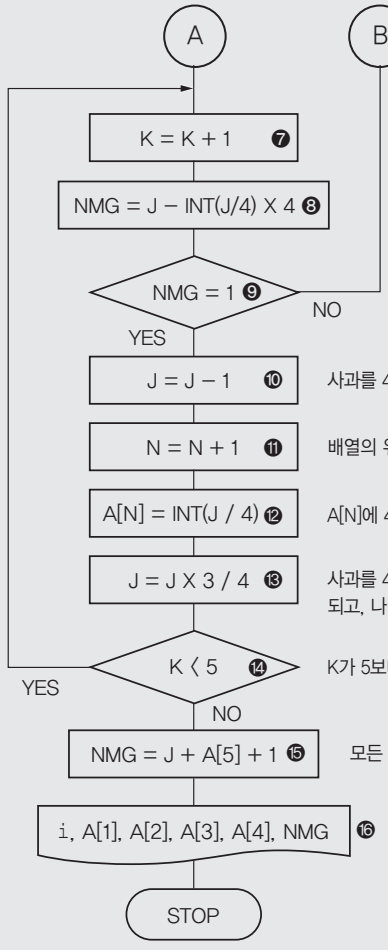
사과의 개수 i는 1씩 누적되며 증가하는 변수이므로 0으로 초기화한다.

❸ N을 0으로 초기화한다.

❹ 사과의 개수 i를 1씩 증가시킨다.

❺ 증가하는 사과의 개수는 변경되면 안 되므로 i를 임시 변수 J에 저장한 후 계산에서는 J를 사용한다.

❻ 분배 횟수 K를 0으로 초기화한다. i가 변경될 때마다 해당 i를 4등분하고 1개가 남는 조건을 5번씩 반복하기 위해 0으로 초기화한다.



❼ 5회의 계산을 수행해야 하므로 K를 1부터 5까지 변화시키기 위해 1씩 증가시킨다.

❽ 사과의 개수를 4등분하고 남은 나머지를 계산한다.

❾ NMG가 1이면 일단 조건에 만족하므로 다음 조건을 검사하기 위해 ❿번으로 가고, 아니면 사과의 개수를 증가시키기 위해 Ⓑ로 간다.

사과를 4등분한 후 남은 1개를 먹었으므로 J에서 1을 차감한다.

배열의 위치 N을 1 증가시킨다.

A[N]에 4등분한 사과 중 한 몫을 저장한다. 즉 한 사람이 가져가야 할 사과의 개수를 저장하는 것이다.

사과를 4등분한 후 자신의 몫에 해당하는 한 몫을 제외한 나머지 사과의 개수를 J에 저장한다. 4/4 중 한 몫은 1/4이 되고, 나머지는 3/4이다.

K가 5보다 작으면 ❼번으로 가고, 아니면 남은 사과의 개수를 계산하기 위해 ⓯번으로 간다.

모든 조건을 만족한 후 남아 있는 사과의 개수를 NMG에 저장시킨다.

조건에 맞는 사과의 총 개수 i, 각 사람이 가져간 사과의 개수, 남아 있는 사과의 개수를 출력하고 끝낸다.

[문제 3] Section 050

① k * i　② sw == 0

변수설명

- **i** : 화폐의 단위가 저장될 변수로, 50000, 10000, 5000, 1000, ..., 1까지 차례로 변경된다.
- **sw** : 화폐 단위 구분을 위한 변수로서 화폐 단위가 50000원, 5000원, 500원, 50원, 5원인지 또는 10000원, 1000원, 100원, 10원, 1원인지를 구분하기 위한 변수로 0 또는 1 을 가진다.
- **j** : 금액이 저장될 변수
- **k** : 화폐의 매수가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int i, sw, j, k;

❷ i = 50000;                    처음에 계산할 화폐 단위는 50000으로 i를 초기화한다.
❸ sw = 0;                       다음에 계산할 화폐 단위는 10000으로, 50000을 5로 나누기 위해 SW를 0으로 초기화한다.
❹ scanf("%d", &j);              금액을 입력받는다.
❺ while (1)                     while 반복문의 시작점이다. 조건이 1이므로 break를 만나기 전까지 ❻~❺번의 문장을 무한 반복한다.
  {
❻   k = j / i;                  C언어에서 정수형 변수는 정수만 저장하므로 정수를 구하는 함수를 사용하지 않아도 된다.
❼   printf("%d, %d\n", i, k);   화폐 단위와 매수를 출력한다.
❽   if (i <= 1)                 i가 1보다 작거나 같으면 매수 계산이 끝난 것이므로 프로그램을 종료하고 아니면 다음 매수 계산을 위해 ❾번으로 간다.
      break;                    반복문(while)을 빠져나가 프로그램을 종료한다.
❾   j -= k*i;                   화폐 매수를 구한 금액을 뺀 나머지 금액을 계산한다.
❿   if (sw == 0)                sw가 0이면 화폐 단위를 5로 나누기 위해 ⓫번으로 가고, 아니면 2로 나누기 위해 ⓮번으로 간다.
    {
⓫     i /= 5;                   10000, 1000, 100, 10, 1에 대한 화폐 단위를 구한다
⓬     sw = 1;                   다음 번에는 2로 나누기 위해 sw를 1로 치환한다.
    }
⓭   else
    {
⓮     i /= 2;                   5000, 500, 50, 5에 대한 화폐 단위를 구한다.
⓯     sw = 0;                   다음 번에는 5로 나누기 위해 sw를 0으로 치환한다.
    }
  }
}
```

[문제 4] Section 042

① nmg != 1　② j / 4

변수설명

- a[5] : 각 사람이 가질 사과의 개수가 저장될 배열
- **i** : 사과의 개수를 셀 변수, 즉 i는 1부터 조건에 맞는 수가 될 때까지 차례로 변경된다.
- **j** : 계산에 사용될 사과의 개수가 저장될 변수, 즉 j는 i를 그대로 받아서 계산한다.
- n : 각 사람이 갖는 사과의 개수를 저장시키기 위한 배열의 위치를 지정해 주는 변수
- k : 분배 횟수를 지정해 주는 변수, 4등분하고 1개가 남는 조건을 5번 반복해야 하므로 k는 1에서 5까지 차례로 변경된다.
- nmg : j를 4로 나눈 나머지와 4등분한 사과를 네 사람이 한 몫씩 가진 후 나머지 사과의 개수가 저장될 변수

```
#include <stdio.h>

main()
{
❶ int i, j, n, k, nmg;
  int a[5];

❷ i = 0;
❸ do                          do~while 반복문의 시작점으로 ❹~⓰번 문장을 반복한다.
  {
❹   n = -1;                   C언어는 배열의 위치를 0부터 시작하기 때문에 배열 a의 첨자가 될 n의 초기값이 순서도와 다르다.
❺   i++;                      사과의 개수 i를 1씩 증가시킨다.
❻   j = i;                    증가하는 사과의 개수는 변경되면 안 되므로 i를 임시 변수 j에 저장한 후 계산에서는 j를 사용한다.
❼   k = 0;                    분배 횟수 k를 0으로 초기화한다. i가 변경될 때마다 해당 i를 4등분하고 1개가 남는 조건을 5번씩 반복하기 위해 0으로 초기화한다.
❽   do                        do~while 반복문의 시작점으로 ❾~⓯번 문장을 반복한다.
    {
❾     k++;                    5회의 계산을 수행해야 하므로 k를 1부터 5까지 변화시키기 위해 1씩 증가시킨다.
❿     nmg = j - j / 4 * 4;    사과의 개수를 4등분하고 남은 나머지를 계산한다. C언어에서 정수형 변수를 나눈 값도 정수이므로 정수를 구하는 함수를 사용하지 않아도 된다.
⓫     if (nmg != 1)           nmg가 1과 같지 않으면 다음 문장을 실행하고, 아니면 ⓬번으로 간다.
        break;                반복문(do~while)을 탈출한다. 제어가 ⓱번으로 이동한다.
⓬     j--;                    사과를 4등분한 후 남은 1개를 먹었으므로 j에서 1을 차감한다.
⓭     n++;                    배열의 위치 n을 1 증가시킨다.
⓮     a[n] = j / 4;           a[n]에 4등분한 사과 중 한 몫을 저장한다. 즉 한 사람이 가져가야 할 사과의 개수를 저장하는 것이다.
⓯     j = j * 3 / 4;          사과를 4등분한 후 자신의 몫에 해당하는 한 몫을 제외한 나머지 사과의 개수를 j에 저장한다. 4/4 중 한 몫은 1/4이 되고, 나머지는 3/4이다.
⓰   } while (k < 5);          k가 5보다 작은 동안 ❾~⓯번을 반복 수행한다.
⓱ } while (nmg != 1 || k < 5); nmg가 1이 아니거나 k가 5보다 작은 동안 ❹~⓰번을 반복 수행한다.
⓲ nmg = j + a[4] + 1;         모든 조건을 만족한 후 남아 있는 사과의 개수를 nmg에 저장시킨다. C언어는 배열의 위치를 0부터 시작하기 때문에 배열 a의 첨자값이 순서도와 다르다.
⓳ printf("%d %d %d %d %d %d", i, a[0], a[1], a[2], a[3], nmg);   조건에 맞는 사과의 총 개수 i, 각 사람이 가져간 사과의 개수, 남아 있는 사과의 개수를 출력하고 끝낸다.

}
```

나는 시험에 나오는 것만 공부한다!

이제 시나공으로 한 번에 정복하세요!

기본서 (필기/실기)

기초 이론부터 완벽하게 공부해서 안전하게 합격하고 싶어요!

특징

자세하고 친절한 이론으로 기초를 쌓은 후 바로 문제풀이를 통해 정리한다.

구성

본권
기출문제(5회)
토막강의

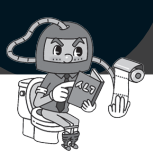
실기
채점 프로그램(워드프로세서, 컴퓨터활용능력, ITQ)

출간종목

컴퓨터활용능력1급 필기/실기
컴퓨터활용능력2급 필기/실기
워드프로세서 필기/실기
정보처리기사 필기/실기
정보처리산업기사 필기
정보처리기능사 필기/실기
사무자동화산업기사 실기
ITQ 엑셀/한글/파워포인트
GTQ 1급/2급

SUMMARY (필기)

핵심이론만 체계적으로 정리한 후 문제풀이를 통해 정리하고 싶어요!



특징

시험에 꼭 나오는 핵심이론으로 개념을 체계적으로 정리한 후 기출문제로 마무리한다.

구성

핵심요약
기출문제(15회)
토막강의

출간종목

컴퓨터활용능력1급 필기
컴퓨터활용능력2급 필기
워드프로세서 필기
정보처리기사 필기
정보처리산업기사 필기
정보처리기능사 필기
사무자동화산업기사 필기

총정리 (필기)

이론은 공부했지만 어떻게 적용되는지 문제풀이를 통해 감각을 익히고 싶어요!

특징

간단하게 이론을 정리한 후 충분한 문제풀이를 통해 실전 감각을 향상시킨다.

구성

핵심요약
기출문제(10회)
모의고사(10회)
토막강의

출간종목

컴퓨터활용능력1급 필기
컴퓨터활용능력2급 필기
워드프로세서 필기
정보처리기사 필기
사무자동화산업기사 필기

기출문제집 (필기/실기)

이론은 완벽해요! 기출문제로 마무리하고 싶어요!

특징

최신 기출문제를 반복 학습하며 최종 마무리한다.

구성

핵심요약(PDF)
기출문제(15회)
토막강의

실기
기출문제(10회)

출간종목

컴퓨터활용능력1급 필기/실기
컴퓨터활용능력2급 필기/실기
워드프로세서 필기
정보처리기사 필기
사무자동화산업기사 필기