시험에 나오는 것만 공부한다! 시나공

원리만 이해하면 쉬운
프로그래밍– 사용자정의함수 9문제
정보처리기사 실기

1. 다음 C언어로 구현된 프로그램을 분석하여 배열 <mines>의 각 칸에 들어갈 값을 쓰시오.

```
#include <stdio.h>
main( ) {
    int field[4][4] = { {0,1,0,1}, {0,0,0,1}, {1,1,1,0}, {0,1,1,1} };
    int mines[4][4] = { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0} };
    int w = 4, h = 4;
    for (int y = 0; y < h; y++) {
        for (int x = 0; x < w; x++) {
            if (field[y][x] == 0) continue;
            for (int j = y - 1; j <= y + 1; j++) {
                for (int i = x - 1; i <= x + 1; i++) {
                    if (chkover(w, h, j, i) == 1)
                        mines[j][i] += 1;
                }
            }
        }
    }
}

int chkover(int w, int h, int j, int i) {
    if (i >= 0 && i < w && j >= 0 && j < h) return 1;
    return 0;
}
```

배열 <field>

0	1	0	1
0	0	0	1
1	1	1	0
0	1	1	1

배열 <mines>

1	1	3	2
3	4	5	3
3	5	6	4
3	5	5	3

[해설]

```
#include <stdio.h>
main( ) {
❶  int field[4][4] = { {0,1,0,1}, {0,0,0,1}, {1,1,1,0}, {0,1,1,1} };
❷  int mines[4][4] = { {0,0,0,0}, {0,0,0,0}, {0,0,0,0}, {0,0,0,0} };
❸  int w = 4, h = 4;
❹  for (int y = 0; y < h; y++) {
❺      for (int x = 0; x < w; x++) {
❻          if (field[y][x] == 0) continue;
❼          for (int j = y - 1; j <= y + 1; j++) {
❽              for (int i = x - 1; i <= x + 1; i++) {
❾⓭                 if (chkover(w, h, j, i) == 1)
⓮                      mines[j][i] += 1;
               }
           }
       }
   }
}

⓾ int chkover(int w, int h, int j, int i) {
⓫     if (i >= 0 && i < w && j >= 0 && j < h) return 1;
⓬     return 0;
  }
```

[알고리즘의 이해]

문제의 코드는 배열 field의 요소가 0이 아닌, 즉 1인 경우, 배열 mines에서 해당 위치를 중심으로 3행 3열의 범위에 1을 더하는 프로그램입니다.

- w와 h는 배열의 행과 열의 길이가 저장된 변수입니다.
- y와 x는 배열 field의 행과 열 위치를 지정해 주는 변수입니다.
- j와 i는 배열 mines에서 1을 더할 범위의 행과 열 위치를 지정해 주는 변수입니다.
- chkover( ) 함수는 j와 i가 배열의 크기를 벗어나는지 검사하는 함수입니다. 벗어났다고 판단되면 0을 반환하여 해당 위치에는 1을 더하지 않도록 합니다.

반복문 수행에 따라 배열 mines에 저장되는 값은 다음과 같습니다.

- 배열 field에서 1의 위치

배열 <field>

0	1	0	1
0	0	0	1
1	1	1	0
0	1	1	1

- 배열 field의 요소가 1인 위치에 대한 배열 mines의 값 변화

배열 <mines>

+1	+1	+1	
+1	+1	+1	

배열 <mines>

1	1	1+1	+1
1	1	1+1	+1

배열 <mines>

1	1	2+1	1+1
1	1	2+1	1+1
		+1	+1

배열 <mines>

	1	1	3	2
	1+1	1+1	3	2
	+1	+1	1	1
	+1	+1		

배열 <mines>

1	1	3	2
2+1	2+1	3+1	2
1+1	1+1	1+1	1
1+1	1+1	+1	

배열 <mines>

1	1	3	2
3	3+1	4+1	2+1
2	2+1	2+1	1+1
2	2+1	1+1	+1

배열 <mines>

1	1	3	2
3	4	5	3
2+1	3+1	3+1	2
2+1	3+1	2+1	1

배열 <mines>

1	1	3	2
3	4	5	3
3	4+1	4+1	2+1
3	4+1	3+1	1+1

배열 <mines>

1	1	3	2
3	4	5	3
3	5	5+1	3+1
3	5	4+1	2+1

배열 <mines>

1	1	3	2
3	4	5	3
3	5	6	4
3	5	5	3

모든 C 프로그램은 반드시 main( ) 함수에서 시작한다.

❶ 4행 4열의 2차원 배열 field를 선언하고 초기화한다.

배열 <field>

0	1	0	1
0	0	0	1
1	1	1	0
0	1	1	1

❷ 4행 4열의 2차원 배열 mines를 선언하고 초기화한다.

배열 <mines>

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

❸ 정수형 변수 w와 h를 선언하고, 각각 4로 초기화한다.
❹ 반복 변수 y가 0부터 1씩 증가하면서 h의 값 4보다 작은 동안 ❺~⓮번을 반복 수행한다.
❺ 반복 변수 x가 0부터 1씩 증가하면서 w의 값 4보다 작은 동안 ❻~⓮번을 반복 수행한다.
❻ filed[y][x]의 값이 0이면 ❼번 이후의 코드를 실행하지 않고 반복문의 처음인 ❺번으로 이동하고, 0이 아니면 ❼번으로 이동한다.
❼ 반복 변수 j가 y-1부터 1씩 증가하면서 y+1보다 작거나 같은 동안 ❽~⓮번을 반복 수행한다.
❽ 반복 변수 i가 x-1부터 1씩 증가하면서 x+1보다 작거나 같은 동안 ❾~⓮번을 반복 수행한다.
❾ w, h, j, i의 값을 인수로 chkover( ) 함수를 호출한 결과가 1이면 ⓮번으로 이동하고, 아니면 ❽번으로 이동한다.
⓾ 정수를 반환하는 chkover( ) 함수의 시작점이다. ❾번에서 전달받은 값을 w, h, j, i가 받는다.
⓫ i와 j가 0보다 크거나 같고, w와 h의 값인 4보다 작으면 함수를 호출했던 ⓭번으로 1을 반환하며 함수를 종료하고, 아니면 ⓬번으로 이동한다.
⓬ 함수를 호출했던 ⓭번으로 0을 반환하고 함수를 종료한다.
⓭ ⓫번에서 1을 반환받았으면 ⓮번으로, ⓬번에서 0을 반환받았으면 ❽번으로 이동한다.
⓮ 'mines[j][i] = mines[j][i] + 1'과 동일하다. mines[j][i]에 1을 누적시킨다.

2. 다음 JAVA로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
public class Test {
    static int[] mkarr( ) {
        int[] tmpArr = new int[4];
        for (int i = 0; i < tmpArr.length; i++)
            tmpArr[i] = i;
        return tmpArr;
    }
    public static void main(String[] args) {
        int[] arr;
        arr = mkarr( );
        for (int i = 0; i < arr.length; i++)
            System.out.print(arr[i]);
    }
}
```

답 : 0123

[해설]

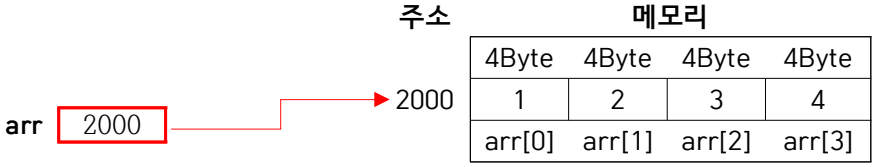
```
public class Test {
❸   static int[] mkarr( ) {
❹       int[] tmpArr = new int[4];
❺       for (int i = 0; i < tmpArr.length; i++)
❻           tmpArr[i] = i;
❼       return tmpArr;
    }
    public static void main(String[] args) {
❶       int[] arr;
❷❽      arr = mkarr( );
❾       for (int i = 0; i < arr.length; i++)
❿           System.out.print(arr[i]);
    }
}
```

모든 Java 프로그램은 반드시 main( ) 함수에서 시작한다.

❶ 정수형 배열 arr을 선언한다. 배열의 요소가 생략되었으므로 배열의 위치를 저장하는 arr만이 메모리에 생성된다.

arr

※ 배열 arr에 4개의 요소(1, 2, 3, 4)를 임의로 초기화하여 선언하면 다음과 같습니다.
(다음 그림에서 arr에 저장된 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했습니다.)

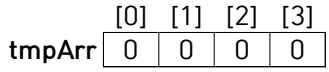


※ ❶번과 같이 요소의 값과 개수를 생략하고 선언만 했다는 것은 arr이 위 그림의 2000과 같은 메모리를 가리키는 주소를 갖지 않은 채 선언만 되었다는 의미입니다. arr은 이후 ❼번 과정에서 정수형 배열의 주소를 전달받아 위와 같이 일반적인 형태를 갖추게 됩니다.

❷ mkarr( ) 메소드를 호출한 후 돌려받은 값을 arr에 저장한다.

❸ 정수형 배열을 반환하는 mkarr( ) 메소드의 시작점이다.

❹ 4개의 요소를 갖는 정수형 배열 tmpArr을 선언한다.



※ Java에서는 배열을 선언하고 초기화하지 않으면 배열의 모든 요소가 0으로 초기화됩니다.

❺ 반복 변수 i가 0부터 1씩 증가하면서 배열 tmpArr의 길이 4보다 작은 동안 ❻번을 반복 수행한다.

- **length** : 배열 클래스의 속성으로 배열 요소의 개수가 저장되어 있음

❻ tmpArr[i]에 i의 값을 저장한다.

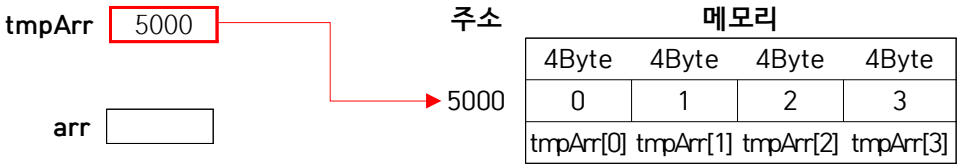
반복문 실행에 따른 변수들의 변화는 다음과 같다.

i	tmpArr [0] [1] [2] [3]
0	0 0 0 0
1	0 1 0 0
2	0 1 2 0
3	0 1 2 3
4	

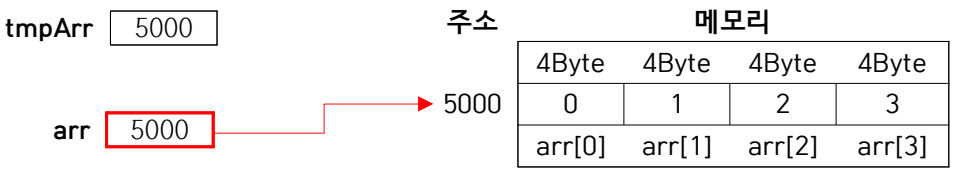
❼ 배열 tmpArr을 반환한다. 배열의 이름을 반환하면 배열의 시작 주소가 반환된다.

❽ ❼번에서 전달받은 배열의 시작 주소를 배열 arr에 저장한다.

※ 배열 tmpArr은 ❸~❻번 과정을 거치면서 다음과 같은 형태를 갖추게 됩니다.
(다음 그림에서 tmpArr이 저장된 주소는 임의로 정한 것이며, 이해를 돕기 위해 10진수로 표현했습니다.)



※ tmpArr이 가진 주소 5000이 ❼~❽번에서 return을 통해 arr에 반환되면서 arr은 tmpArr과 같은 주소를 갖는 배열이 됩니다.



❾ 반복 변수 i가 0부터 1씩 증가하면서 배열 arr의 길이 4보다 작은 동안 ❿번을 반복 수행한다.

❿ arr[i]의 값을 출력한다.

반복문 실행에 따른 변수들의 변화는 다음과 같다.

i	arr[i]	출력
0	0	**0**
1	1	0**1**
2	2	01**2**
3	3	012**3**
4		

3. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
#include <stdio.h>
int isPrime(int number) {
    for (int i = 2; i < number; i++)
        if (number % i == 0) return 0;
    return 1;
}

int main() {
    int number = 13195;
    int max_div = 0;
    for (int i = 2; i < number; i++)
        if (isPrime(i) == 1 && number % i == 0) max_div = i;
    printf("%d", max_div);
}
```

답 : 29

[해설]

```
   #include <stdio.h>
❺ int isPrime(int number) {
❻     for (int i = 2; i < number; i++)
❼         if (number % i == 0) return 0;
❽     return 1;
   }

   int main() {
❶      int number = 13195;
❷      int max_div = 0;
❸      for (int i = 2; i < number; i++)
❹❾         if (isPrime(i) == 1 && number % i == 0) max_div = i;
⓾      printf("%d", max_div);
}
```

모든 C언어 프로그램은 반드시 main( ) 함수에서 시작한다.

❶ 정수형 변수 number를 선언하고 13195로 초기화한다.

❷ 정수형 변수 max_div를 선언하고 0으로 초기화한다.

❸ 반복 변수 i가 2부터 1씩 증가하면서 number보다 작은 동안 ❹번을 반복 수행한다.

첫 번째 반복

❹ i의 값 2를 인수로 isPrime을 호출한 결과가 1이고 number를 i로 나눈 나머지가 0이면 max_div에 i의 값을 저장한다.

❺ 정수를 반환하는 isPrime( ) 함수의 시작점이다. ❹번에서 전달받은 2를 number가 받는다.

❻ 반복 변수 i가 2부터 1씩 증가하면서 2보다 작은 동안 ❼번을 반복 수행한다. i의 값 2는 2보다 작지 않으므로 ❼번을 수행하지 않고 ❽번으로 이동한다.
❽ 1을 반환하면서 함수를 호출했던 ❾번으로 이동한다.
❾ ❽번에서 돌려받은 값은 1이지만, number의 값 13195를 i의 값 2로 나눈 나머지는 1이므로 **max_div = i**를 수행하지 않고 ❸번으로 돌아가 i의 값을 1 증가시킨다.

두 번째 반복

❹ i의 값 3을 인수로 isPrime을 호출한 결과가 1이고 number를 i로 나눈 나머지가 0이면 max_div에 i의 값을 저장한다.
❺ ❹번에서 전달받은 3을 number가 받는다.
❻ 반복 변수 i가 2부터 1씩 증가하면서 3보다 작은 동안 ❼번을 반복 수행한다.
❼ 3을 i로 나눈 나머지가 0이면 0을 반환하면서 함수를 호출했던 ❾번으로 이동한다.
❻~❼번 반복문 실행에 따른 변수들의 변화는 다음과 같다.

number	i
3	2
	3

❽ 1을 반환하고 함수를 호출했던 ❾번으로 이동한다.
❾ ❽번에서 돌려받은 값은 1이지만, number의 값 13195를 i의 값 3으로 나눈 나머지는 1이므로 **max_div = i**를 수행하지 않고 ❸번으로 돌아가 i의 값을 1 증가시킨다.

세 번째 반복

❹ i의 값 4를 인수로 isPrime을 호출한 결과가 1이고 number를 i로 나눈 나머지가 0이면 max_div에 i의 값을 저장한다.
❺ ❹번에서 전달받은 4를 number가 받는다.
❻ 반복 변수 i가 2부터 1씩 증가하면서 4보다 작은 동안 ❼번을 반복 수행한다.
❼ 4를 i로 나눈 나머지가 0이면 0을 반환하면서 함수를 호출했던 ❾번으로 이동한다.
❻~❼번 반복문 실행에 따른 변수들의 변화는 다음과 같다.

number	i
4	2

❾ ❽번에서 돌려받은 값이 0이고, number의 값 13195를 i의 값 4로 나눈 나머지는 3이므로 **max_div = i**를 수행하지 않고 ❸번으로 돌아가 i의 값을 1 증가시킨다.

네 번째 반복

❹ i의 값 5를 인수로 isPrime을 호출한 결과가 1이고 number를 i로 나눈 나머지가 0이면 max_div에 i의 값을 저장한다.
❺ ❹번에서 전달받은 5를 number가 받는다.
❻ 반복 변수 i가 2부터 1씩 증가하면서 5보다 작은 동안 ❼번을 반복 수행한다.
❼ 5를 i로 나눈 나머지가 0이면 0을 반환하고 함수를 호출했던 ❾번으로 이동한다.
❻~❼번 반복문 실행에 따른 변수들의 변화는 다음과 같다.

number	i
5	2
	3
	4
	5

❽ 1을 반환하고 함수를 호출했던 ❾번으로 이동한다.

❾ ❽번에서 돌려받은 값이 1이고, number의 값 13195를 i의 값 5로 나눈 나머지도 0이므로 max_div에 5를 저장한 후 ❸번으로 돌아가 i의 값을 1 증가시킨다.

위의 과정을 통해 다음 사항들을 알 수 있다.

- isPrime( ) 함수는 인수를 2에서 시작하여 전달받은 수보다 1 작을 때까지 나눴을 때 끝까지 나머지가 0이 아니면 1을 반환하는 것으로 보아 소수를 찾는 함수임을 알 수 있다.
- ❾번에서 isPrime(i)가 1이라는 것은 i가 소수임을 의미하고, number를 i로 나눈 나머지가 0이라는 것은 i가 number의 약수라는 의미이므로, max_div에는 소수이자 number의 약수인 수가 저장된다.
- i의 값이 1씩 증가하면서 number보다 1 작을 때까지 위 과정을 수행하므로 number의 소수로 된 약수 중 가장 큰 소수에 해당하는 값이 max_div에 저장된다.
- 13195의 소수로 된 약수는 5, 7, 13, 29이며, 이 중 가장 큰 소수인 29가 최종적으로 max_div에 저장된다.
- 자세한 값의 변화는 다음 표를 통해 확인하자.

main() 함수			isPrime() 함수		
number	i	max_div	number	i	반환값
13195	2	0	2	2	1
⋮	3		3	2	1
⋮				3	
	4		4	2	0
	5	5	5	2	1
				3	
				4	
				5	
	⋮	⋮	⋮	⋮	⋮
	29	29	29	2	1
				3	
				4	
				⋮	
				28	
				29	
	⋮	⋮	⋮	⋮	⋮
	13194		13194	2	0
	13195				

❿ max_div의 값 **29**를 정수로 출력한다.

결과 29

4. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
#include <stdio.h>
main() {
    int res = mp(2, 10);
    printf("%d", res);
}
int mp(int base, int exp) {
    int res = 1;
    for (int i = 0; i < exp; i++)
        res *= base;
    return res;
}
```

답 : 1024

[해설]

```
❶❼  #include <stdio.h>
     main() {
❶❼      int res = mp(2, 10);
❽       printf("%d", res);
     }
❷ int mp(int base, int exp) {
❸       int res = 1;
❹       for (int i = 0; i < exp; i++)
❺           res *= base;
❻       return res;
     }
```

모든 C언어 프로그램은 반드시 main( ) 함수에서 시작한다

❶ 정수형 변수 res를 선언하고, 2와 10을 인수로 mp( ) 함수를 호출한 후 돌려받은 값을 res에 저장한다.

❷ 정수를 반환하는 mp( ) 함수의 시작점이다. ❶번에서 전달받은 2와 10을 base와 exp가 받는다.

❸ 정수형 변수 res를 선언하고 1로 초기화한다.

❹ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 10보다 작은 동안 ❺번을 반복 수행한다.

❺ 'res = res * base;'와 동일하다. res와 base를 곱한 값을 res에 저장한다.

반복문 실행에 따른 변수들의 값의 변화는 다음과 같다.

base	exp	i	res
2	10		1
		0	2
		1	4
		2	8
		3	16

		4	32
		5	64
		6	128
		7	256
		8	512
		9	1024
		10	

❻ res의 값 1024를 mp( )를 호출했던 ❼번으로 반환한다.
❼ ❻번으로부터 반환받은 값 1024를 res에 저장한다.
❽ res의 값을 출력한다.

결과 **1024**

5. 다음 Java로 구현된 프로그램을 분석하여 괄호에 들어갈 알맞은 예약어를 쓰시오.

```
public class Test {
    public static void main(String[] args) {
        System.out.print(Test.check(1));
    }
    (      ) String check(int num){
        return (num >= 0) ? "positive" : "negative";
    }
}
```

답 : static

[해설]

static은 클래스 이름으로 메소드에 접근하기 위해 사용하는 예약어로, 메소드를 사용하기 위해서는 메소드가 포함된 클래스의 객체 변수를 선언한 후 **[객체 변수].[메소드]**의 방식으로 접근해야 하지만 static을 이용하면 객체 변수 없이 **[클래스 이름].[메소드]**의 방식으로 접근하는 것이 가능해집니다.

```
public class Test {
    public static void main(String[] args) {
❶❹     System.out.print(Test.check(1));
    }
❷   static String check(int num){
❸       return (num >= 0) ? "positive" : "negative";
    }
}
```

모든 Java 프로그램은 반드시 main( ) 메소드에서 시작한다.

❶ 1을 인수로 Test 클래스의 check( ) 메소드를 호출한 후 돌려받은 값을 출력한다.

❷ 문자열을 반환하는 check( ) 메소드의 시작점이다. ❶번에서 전달받은 1을 num이 받는다.

❸ num이 0보다 크거나 같으면 "positive"를 반환하고, 아니면 "negative"를 반환한다. num의 값 1은 0보다 크므로 "positive"를 ❹번으로 반환한다.

❹ ❸번에서 돌려받은 positive를 출력한다.

결과 positive

6. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
#include <stdio.h>
int r1() {
        return 4;
}
int r10() {
        return (30 + r1());
}
int r100() {
        return (200 + r10());
}
int main() {
        printf("%d\n", r100());
        return 0;
}
```

234

답 : 234

[해설]

```
#include <stdio.h>
int r1() {
    ❹   return 4;
}                          4 반환
int r10() {
    ❸❺ return (30 + r1());
}                                  34(30 + 4) 반환
int r100() {
    ❷❻ return (200 + r10());
}                                          234(200 + 34) 반환
int main() {
    ❶❼ printf("%d\n", r100());
    ❽   return 0;
}
```

❶ 인수 없이 r100( ) 함수를 호출한 다음 반환받은 값을 화면에 출력한다. ❷번으로 이동한다.
❷ r10( ) 함수를 호출하여 반환받은 값에 200을 더한 후 r100( ) 함수를 호출한 ❶번으로 반환한다. r10( ) 함수를 호출하기 위해 ❸번으로 이동한다.
❸ r1( ) 함수를 호출하여 반환받은 값에 30을 더한 후 r10( ) 함수를 호출한 ❷번으로 반환한다. r1( ) 함수를 호출하기 위해 ❹번으로 이동한다.
❹ 반환값 4를 가지고 r1( ) 함수를 호출했던 ❺번으로 이동한다.
❺ ❹번에서 반환받은 값 4에 30을 더한 34를 가지고 r10( ) 함수를 호출했던 ❻번으로 이동한다.
❻ ❺번에서 반환받은 값 34에 200을 더한 234를 가지고 r100( ) 함수를 호출했던 ❼번으로 이동한다.
❼ ❻번에서 반환받은 값 234를 정수로 출력하고 커서를 다음 줄로 이동한다.
결과 234
❽ main( ) 함수에서의 'return 0'은 프로그램의 종료를 의미한다.

7. 다음 Java로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
public class Test {
    static int[ ] arr( ) {
        int a[ ] = new int[4];
        int b = a.length;
        for(int i = 0; i < b; i++)
            a[i] = i;
        return a;
    }

    public static void main(String[ ] args) {
        int a[ ] = arr( );
        for(int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
    }
}
```

답 : 0 1 2 3

[해설]

모든 Java 프로그램은 반드시 main( ) 메소드부터 시작해야 한다.

```
public static void main(String[] args) {
❶   int a[] = arr();
    for(int i = 0; i < a.length; i++)
        System.out.print(a[i] + " ");
}
```

❶ 정수형 배열 a를 선언하고, 인수 없이 arr( ) 메소드를 호출한 다음 돌려받은 값을 배열 a에 저장한다.

```
❷ static int[] arr() {
❸   int a[] = new int[4];
❹   int b = a.length;
❺   for(int i = 0; i < b; i++)
❻       a[i] = i;
❼   return a;
  }
```

❷ 메소드의 리턴값이 정수형 배열인 arr( ) 메소드의 시작점이다.

❸ 4개의 요소를 갖는 정수형 배열 a를 선언한다.

❹ 정수형 변수 b를 선언하고 배열 a의 길이인 4로 초기화한다.

- **length** : 배열 클래스의 속성으로 배열 요소의 개수가 저장되어 있다. a 배열은 4개의 요소를 가지므로 a.length는 4를 가지고 있다.

❺ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 b보다 작은 동안 ❻번 문장을 반복 수행한다.

❻ a[i]에 반복 변수 i의 값을 저장한다.

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

❼ 배열 a에 저장된 값들을 호출한 곳(main( ) 메소드)으로 반환한다.

```
public static void main(String[] args) {
❶❽ int a[] = arr();
❾   for(int i = 0; i < a.length; i++)
❿       System.out.print(a[i] + " ");
}
```

❽ arr( ) 메소드로부터 반환받은 값들을 main( ) 메소드의 배열 a에 저장한다.

❾ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 a 배열의 길이 4보다 작은 동안 ❿번 문장을 반복 수행한다.

❿ a[i]의 값과 공백 한 칸을 출력한다.

8. 다음 C언어로 구현된 프로그램을 분석하여 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
#include <stdio.h>
void align(int a[ ]) {
    int temp;
    for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4 - i; j++)
            if (a[j] > a[j+1]) {
                temp = a[j];
                a[j] = a[j+1];
                a[j+1] = temp;
            }
}

main( ) {
    int a[ ] = { 85, 75, 50, 100, 95 };
    align(a);
    for (int i = 0; i < 5; i++)
        printf("%d ", a[i]);
}
```

답 : 50 75 85 95 100

[해설]

문제의 코드는 버블 정렬 알고리즘을 이용하여 배열 a에 저장된 값을 오름차순으로 정렬한 후 출력하는 프로그램이다. 버블 정렬 알고리즘은 첫 번째 자료와 두 번째 자료를, 두 번째 자료와 세 번째 자료를, 세 번째와 네 번째를, …, 이런 식으로 (마지막-1)번째 자료와 마지막 자료를 비교하여 교환하면서 자료를 정렬한다. 1회전을 수행하고 나면 가장 큰 자료가 맨 뒤로 이동하므로 2회전에서는 맨 끝에 있는 자료는 정렬에서 제외되고, 2회전을 수행하고 나면 끝에서 두 번째 자료까지는 정렬에서 제외된다. 이렇게 정렬을 1회전 수행할 때마다 정렬에서 제외되는 데이터가 하나씩 늘어난다.

모든 C 프로그램은 반드시 main( ) 함수부터 시작해야 한다.

```
main( ) {
❶ int a[] = { 85, 75, 50, 100, 95 };
❷ align(a);
   for (int i = 0; i < 5; i++)
      printf("%d ", a[i]);
}
```

❶ 배열을 선언할 때 사용할 개수를 정하지 않으면 초기값으로 지정된 수만큼 배열의 요소가 만들어진다.

	a[0]	a[1]	a[2]	a[3]	a[4]
배열 a	85	75	50	100	95

❷ a를 인수로 하여 함수 align( )을 호출한다. 인수로 배열의 이름을 지정하면 배열의 시작 주소가 인수로 전달된다. 그러니까 align(a)는 align(&a[0])과 같은 의미다.

```
❸ void align(int a[]) {
❹ int temp;
❺  for (int i = 0; i < 4; i++)
❻      for (int j = 0; j < 4 - i; j++)
❼         if (a[j] > a[j+1]) {
❽              temp = a[j];
❾              a[j] = a[j+1];
❿              a[j+1] = temp;
⓫         }
⓬ }
```

❸ 반환값이 없는 align() 함수의 시작점이다. ❷에서 'align(a)'라고 했으므로 정수형 배열 a는 main( ) 함수의 a 배열의 시작 주소를 받는다.

	a[0]	a[1]	a[2]	a[3]	a[4]
배열 a	85	75	50	100	95

❹ 정수형 변수 temp를 선언한다.

❺ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 4보다 작은 동안 ❻번을 반복하여 수행한다.

❻ 반복 변수 j가 0에서 시작하여 1씩 증가하면서 4-i보다 작은 동안 ❼번을 반복하여 수행한다.

❼ a[j]가 a[j+1]보다 크면 ❽~❿ 사이의 문장을 실행한다.

반복문 실행에 따른 변수들의 변화는 다음과 같다.

i	j	a[j]	a[j+1]	temp	배열 a
0	0 1 2 3 4	85 75 85 50 85 100 95	75 85 50 85 100 95 100	85 85 100	a[0] a[1] a[2] a[3] a[4] ~~85~~ ~~75~~ ~~50~~ ~~100~~ ~~95~~ 75 ~~85~~ 85 95 100 50
1	0 1 2 3	75 50 75 85	50 75 85 95	75	a[0] a[1] a[2] a[3] a[4] ~~75~~ ~~50~~ 85 95 100 50 75
2	0 1 2	50 75	75 85		a[0] a[1] a[2] a[3] a[4] 50 75 85 95 100
3	0 1	50	75		a[0] a[1] a[2] a[3] a[4] 50 75 85 95 100
4					

⓫ if문의 끝이다.

⓬ 함수를 마치고 align(a) 함수를 호출했던 main( ) 함수로 제어를 옮긴다. a배열의 주소를 받아서 처리했으므로 main( ) 함수에 있는 a배열에 정렬의 결과가 그대로 반영되어 있다. 즉 정렬 결과를 반환할 필요가 없다.

```
main( ) {
   int a[] = { 85, 75, 50, 100, 95 };
   align(a);
⓭ for (int i = 0; i < 5; i++)
⓮    printf("%d ", a[i]);
}
```

⓭ 반복 변수 i가 0에서 시작하여 1씩 증가하면서 5보다 작은 동안 ⓮번을 반복 수행한 후 프로그램을 종료한다.

⓮ a[i]의 값을 정수로 출력한 후 한 칸을 띄운다.

결과 50 75 85 95 100

※ ❷번에서 인수로 배열의 이름, 즉 배열의 시작 주소가 전달되었으므로 호출된 align( ) 함수에서의 값의 변화는 main( ) 메소드의 a 배열에도 영향을 줍니다. 그러므로 출력되는 a 배열의 값은 정렬이 수행된 후의 값이 출력됩니다.

9. 다음 C언어로 구현된 프로그램을 분석하여 5를 입력했을 때 그 실행 결과를 쓰시오. (단, 출력문의 출력 서식을 준수하시오.)

```
#include <stdio.h>
int func(int a) {
    if (a <= 1) return 1;
    return a * func(a - 1);
}
int main() {
    int a;
    scanf("%d", &a);
    printf("%d", func(a));
}
```

5 × 4 × 3 × 2 = 120

답 : 120

[해설]

모든 C언어 프로그램은 반드시 main( ) 함수에서 시작한다.

```
int main() {
❶   int a;
❷   scanf("%d", &a);
❸   printf("%d", func(a));
}
```

❶ 정수형 변수 a를 선언한다.

❷ 정수를 입력받아 a에 저장한다. 5가 입력되었다고 가정하였으므로 a에는 5가 저장된다.

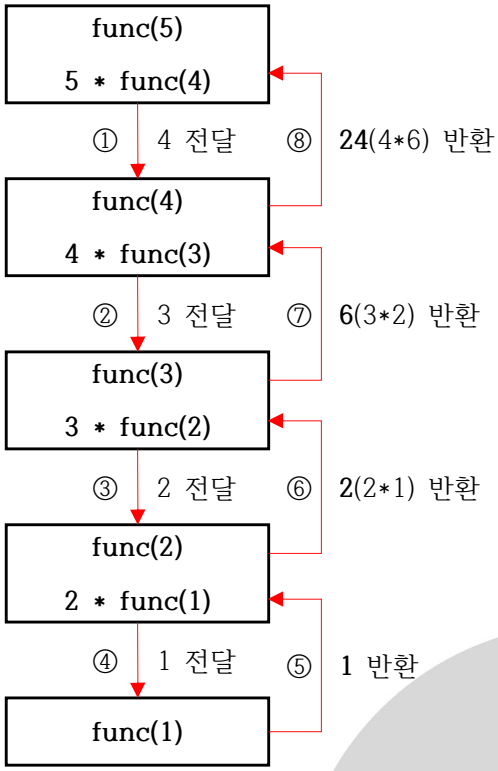
❸ a의 값 5를 인수로 하여 func( ) 함수를 호출한 후 돌려받은 값을 정수로 출력한다.

```
❹ int func(int a) {
❺   if (a <= 1) return 1;
❻   return a * func(a - 1);
}
```

❹ 정수를 반환하는 func( ) 함수의 시작점이다. ❸번에서 전달받은 5를 a가 받는다.

❺ a가 1보다 작거나 같으면 함수를 호출했던 곳으로 1을 반환하고 함수를 종료한다. a의 값 5는 1보다 작거나 같지 않으므로 ❻번으로 이동한다.

❻ a-1을 인수로 하여 func( ) 함수를 호출한 후 돌려받은 값과 a를 곱하여 함수를 호출했던 곳으로 반환하고 함수를 종료한다. a가 1보다 큰 동안 자신을 호출하는 과정이 수행되다 a가 1보다 작거나 같아지면 1이 반환되면서 호출했던 과정을 복귀한다. 이 때 반환된 값은 먼저 호출된 func( ) 함수에 반환할 값으로 계산된다는 것을 염두에 두고 전체 과정을 개괄적으로 살펴보자.



a-1의 값 4를 인수로 func( ) 함수를 호출한 후 돌려받은 값이 24이므로 a의 값 5를 곱한 값 120을 func(5)를 호출했던 ❼번으로 반환한다.

```
   int main() {
❶      int a;
❷      scanf("%d", &a);
❸❼    printf("%d", func(a));
   }
```

❼ ❻번에서 돌려받은 값 120을 정수로 출력한다.

결과 120