01-2 컴파일과 컴파일러 사용법

1. 답 소스 파일 작성 – 전처리 – 컴파일 – 링크

2. 답 ① 기계어 ② 개체 파일 ③ 링크

02-1 C 프로그램의 구조와 데이터 출력 방법

1. 답 ③

주석문은 함수와 관계없이 소스 코드 어디서나 사용할 수 있습니다.

2. 답 ① %d ② %d ③ %lf

3. 답

실행결과

```
Happy!
Baby
```

printf("Hello world!\n"); 문장은 주석기호 /* */ 안에 있으므로 주석문이 됩니다. Be가 출력된 후에 \r에 의해 커서가 B의 위치로 이동해 Happy!가 출력되므로 Be가 지워집니다. 그 후에 \n에 의해 줄이 바뀌고 Baby가 출력됩니다.

02-2 상수와 데이터 표현 방법

1. 답

10진수	8진수	16진수	2진수
11	13	B	1011
17	21	11	10001
26	32	1A	11010
65	101	41	1000001

2. 답 -10 (정) 1e4 (실) -1. (실) -1.5e-3 (실)

+032 (정) 3.14 (실) 0xff (정)

3.

```
#include <stdio.h>
int main(void)
{
    printf("학번 : %d\n", 32165);            // 정수는 %d로 출력
    printf("이름 : %s\n", "홍길동");          // 문자열은 %s로 출력
    printf("학점 : %c\n", 'A');              // 문자는 %c로 출력
    return 0;
}
```

03-1 변수

1. 답 float, double

unsigned int형에서 int를 생략하고 unsigned만 사용할 수도 있습니다.

2. 답

실행결과

```
a : 6
```

a = a + 1;은 현재의 a의 값에 1을 더해 다시 a에 대입하는 문장입니다. 따라서 1부터 3까지 계속 더하면 6이 됩니다.

3.

```
#include <stdio.h>

int main(void)
{
    int kor = 70, eng = 80, mat = 90;        // 세 과목의 변수 선언과 초기화
    int tot;                                 // 총점을 저장할 변수 선언

    tot = kor + eng + mat;                   // 세 변수의 값을 더해 총점 변수에 저장
    printf("국어 : %d, 영어 : %d, 수학 : %d\n", kor, eng, mat);         // 점수 출력
    printf("총점 : %d", tot);                // 총점 출력

    return 0;
}
```

03-2 데이터 입력

1. 답 ③

① ch는 char형 변수이므로 %c 변환 문자를 사용합니다.

② sh는 short형 변수이므로 %hd 변환 문자를 사용합니다.

③ 2개 이상의 변수에 입력할 때는 변환 문자를 이어서 사용합니다.

④ double형 변수에 입력할 때는 %lf 변환 문자를 사용합니다.

2. 답 ① fruit ② &cnt

char 배열에 문자열을 입력할 때는 배열명만 사용하고 그 외의 변수에는 &를 씁니다.

3.

```
#include <stdio.h>

int main(void)
{
    char ch;                                // 문자를 저장할 변수

    printf("문자 입력 : ");                  // 입력 안내 메시지
    scanf("%c", &ch);                       // 변수 ch에 문자 입력
    printf("%c문자의 아스키 코드 값은 %d입니다.", ch, ch);
    // ch의 값을 변환 문자를 바꿔 한 번은 문자로 한 번은 아스키 코드 값으로 출력한다.
    return 0;
}
```

04-1 산술 연산자, 관계 연산자, 논리 연산자

1.

```
#include <stdio.h>

int main(void)
{
    double a = 4.0, b = 1.2;

    printf("%.1lf + %.1lf = %.1lf\n", a, b, a + b);
    printf("%.1lf - %.1lf = %.1lf\n", a, b, a - b);
    printf("%.1lf * %.1lf = %.1lf\n", a, b, a * b);
```

```
    printf("%.1lf / %.1lf = %.1lf\n", a, b, a / b);
    // double형이므로 소수점까지 계산한다.
    // 소수점 이하 첫째 자리까지 출력해야 하므로 % 다음에 .1을 사용한다.

    return 0;
}
```

2.

```
#include <stdio.h>

int main(void)
{
     int a, b, tot;
     double avg;

     printf("두 과목의 점수 : ");
     scanf("%d%d", &a, &b);          // 점수 입력
     tot = a + b;                    // 두 점수를 더해 총점을 구한다.
     avg = tot / 2.0;
     // 평균 계산, tot가 정수형이므로 나누는 값을 2.0과 같이 실수값으로 사용해야 한다.
     // 그렇지 않으면 피연산자가 모두 정수값이므로 몫을 계산한다.

     printf("평균 : %.1lf\n", avg);

     return 0;
}
```

3.

```
#include <stdio.h>

int main(void)
{
     int kor = 3, eng = 5, mat = 4;                  // 국어, 영어, 수학의 학점 초기화
     int credits;                                    // 전체 학점을 저장할 변수
     int res;                                        // 연산 결과를 저장할 변수
     double kscore = 3.8, escore = 4.4, mscore = 3.9;  // 각 과목의 평점 초기화
     double grade;                                   // 평점의 평균을 저장할 변수

     credits = kor + eng + mat;                      // 전체 학점 계산
     grade = (kscore + escore + mscore) / 3.0;       // 평점의 평균 계산
```

```
    res = (credits >= 10) && (grade > 4.0);          // 전체 학점이 10학점 이상이고
    // 평점 평균이 4.0보다 크면 참이므로 결과는 1, 그렇지 않으면 거짓이므로 결과는 0
    printf("%d\n", res);

    return 0;
}
```

04-2 그 외 유용한 연산자

1.

```
#include <stdio.h>

int main(void)
{
    int res;

    res = sizeof (short) > sizeof (long);
    // sizeof의 피연산자로 자료형 이름을 사용해 크기를 바이트 단위로 계산한다.
    // short형의 크기가 long형보다 크면 참이므로 1, 그렇지 않으면 0을 res에 저장한다.

    printf("%s\n", (res == 1) ? "short" : "long");
    // res가 1과 같으면 short형의 크기가 크므로 출력하고
    // 그렇지 않으면 long형을 출력한다.

    return 0;
}
```

2.

```
#include <stdio.h>

int main(void)
{
    int seats = 70;         // 경기장의 좌석 수(seats) 초기화
    int audience = 65;      // 입장객 수(audience) 초기화
    double rate;            // 입장률(rate)을 저장할 변수

    rate = (double)audience / (double)seats * 100.0;
    // 'audience / seats'를 바로 연산하면 둘 다 int형이므로 몫을 계산한다.
    // 이 경우 입장객 수가 좌석 수보다 크지 않으므로 항상 0이 출력된다.
```

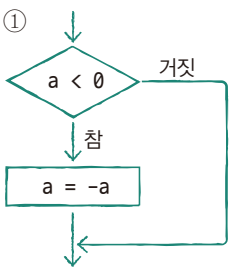
```
    // 따라서 소수점까지 계산할 수 있도록 double형으로 형 변환한다.
    // 나누기(/)와 곱하기(*)는 우선순위가 같으므로 연산 방향에 따라 왼쪽부터
    // 나누기 연산이 먼저 수행된다.

    printf("입장률 : %.1lf%%\n", rate);              // 입장률 출력
                                  %는 변환 문자와 함께 사용되므로
    return 0;                     % 자체를 출력할 때는 이처럼 두 번 연속 사용해야 합니다.
}
```

3.

```
#include <stdio.h>

int main(void)
{
    int hour, min, sec;     // 시, 분, 초를 저장할 변수
    double time = 3.76;     // 시간 초기화

    hour = (int) time;      // 형 변환으로 정수 부분만을 골라낸다.
    time -= hour;           // 한 시간이 안 되는 부분만을 다시 저장한다.
    time *= 60.0;           // 분 단위로 환산
    min = (int) time;       // 정수 부분만을 골라내어 분으로 저장한다.
    time -= min;            // 일분이 안 되는 부분만을 다시 저장한다.
    time *= 60.0;           // 초 단위로 환산
    sec = (int) time;       // 정수 부분만을 골라내어 초로 저장한다.
    printf("3.76시간은 %d시간 %d분 %d초입니다.\n", hour, min, sec);  // 변환한 시간 출력

    return 0;
}
```

05-1 if문

1. ①

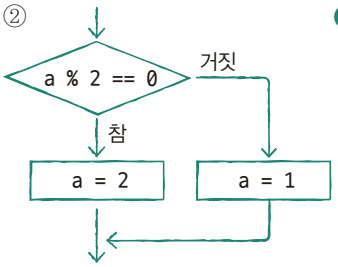


답

```
if(a < 0)           // a가 0보다 작으면
{
    a = -a;         // a의 부호를 바꿔 a에 대입
}
```

a의 값이 0보다 크거나 같으면 조건식이 거짓이므로 a의 부호를 바꾸지 않습니다. 즉, if문을 사용해 a의 값이 음수일 때 양수로 바꾸는 코드입니다.

②



답

```
if((a % 2) == 0)  // a를 2로 나눈 나머지가 0이면
{
    a = 2;        // a에 2를 대입
}
else              // a를 2로 나눈 나머지가 0이 아니면
{
    a = 1;        // a에 1을 대입
}
```

a를 2로 나눈 나머지가 0이면 a가 짝수고 그렇지 않으면 홀수입니다. 즉, a가 짝수면 2로 바꾸고 홀수면 1로 바꾸는 코드입니다.

2. 답 ① chest <= 90 ② chest <= 100

②의 조건식으로 (chest > 90) && (chest <= 100)를 사용할 수도 있지만 ①의 조건이 거짓인 경우 ②의조건을 검사하므로 chest > 90의 조건을 넣을 필요가 없습니다.

3.

```
#include <stdio.h>
int main(void)
{
    double height = 179.5;                    // 키를 저장할 변수 선언과 초기화
    double weight = 75.0;                     // 몸무게를 저장할 변수 선언과 초기화
    if((height >= 187.5) && (weight < 80.0))  // 키가 187.5 이상이고
    {                                         // 몸무게가 80 미만이면
        printf("ok\n");                       // ok 출력
    }
    else                                      // 그 이외의 경우
    {
        printf("cancel\n");                   // cancel 출력
    }
    return 0;
}
```

if문의 조건식에서 관계 연산자(>=, <)가 논리 연산자(&&)보다 우선순위가 높아 관계 연산자가 먼저 수행되므로 조건식 안에 괄호가 없어도 결과는 같습니다. 그러나 연산자 우선순위를 명

확히 하기 위해 괄호를 사용하는 것이 좋습니다. 논리곱 연산자는 두 피연산자가 모두 참인 경우만 참이므로 키가 187.5보다 작거나 몸무게가 80 이상이면 조건식은 거짓이 됩니다.

05-2 if문 활용과 switch ~ case문

1. 답 ③

break는 switch ~ case문의 블록을 탈출할 때 사용됩니다.

2. 답 default:

조건식의 결과가 0이 아닌 경우는 모두 참이 출력되므로 default를 사용합니다.

3.

```
#include <stdio.h>
int main(void)
{
    int age = 25, chest = 95;
    char size;
    if (age < 20)                           // 나이가 20 미만이면
    {
        if (chest < 85) size = 'S';         // 20세 미만 chest 값에 따라 사이즈를 결정
        else if (chest < 95) size = 'M';
        else size = 'L';
    }
    else                                    // 나이가 20보다 크거나 같으면
    {
        if (chest < 90) size = 'S';         // 20세 이상 기준으로 사이즈를 결정
        else if (chest < 100) size = 'M';
        else size = 'L';
    }
    printf("사이즈는 %c입니다.\n", size);      // 결정된 사이즈 출력
    return 0;
}
```

06-1 while문, for문, do ~ while문

1. 답

```
do {                            // 중괄호를 do 옆에 붙이면 한 줄을 줄일 수 있다.
    scanf("%d", &a);            // 키보드로 a에 정수 입력
} while (a < 0);                // 입력된 값이 음수면 입력을 반복
```

반복할 문장을 실행한 후 조건을 검사하므로 do ~ while문으로 작성합니다. 입력되는 값이 음수면 다시 입력하고 0을 포함해 양수가 입력되면 반복을 끝냅니다. 프로그램에서 양수가 필요한 경우 잘못된 입력을 걸러 내는 용도로 이 코드를 사용할 수 있습니다.

2. 답

```
for (i = 0; i < 5; i++)         // 변수 i를 하나씩 증가시키면서 5보다 작은 동안 반복
{
    printf("Be happy\n");       // 실행문
}
```

i = 0은 초기식으로 한 번 실행되고 출력문과 증감식이 반복되므로 for문으로 작성합니다. 또는 증감식을 반복할 문장으로 포함시켜 while문으로 작성하는 것도 가능합니다.

```
i = 0;                          // 일단 반복문 시작 전에 i를 0으로 초기화
while (i < 5)                   // i가 5보다 작은 동안 반복
{
    printf("Be happy\n");       // 실행문
    i++;                        // 여기서 i값을 하나 증가시킨다.
}
```

3.

```
#include <stdio.h>
int main(void)
{
    int i;
    for (i = 0; i < 10; i++)        // i는 0부터 9까지 하나씩 증가되므로 열 번 반복
    {
        printf("$");                // 실행문
    }
    return 0;
}
```

06-2 반복문 활용

1. 답 12번

바깥쪽 i-for문이 세 번 반복되고 안쪽 j-for문이 네 번 반복되므로 j-for문 안에 있는 문장은 총 열두 번 반복됩니다.

2. 답 9번

안쪽의 j-for문은 기본적으로 네 번 반복되나 j가 2일 때 break에 의해 반복이 끝나므로 결국 출력문은 j의 값이 0, 1, 2일 때 세 번만 실행됩니다. 이 반복문을 바깥쪽 i-for문에서 다시 세 번 반복하므로 출력문은 총 아홉 번 실행됩니다. break는 가장 가까운 반복문인 j-for문만 벗어나며 바깥쪽 i-for문은 계속 반복됩니다.

3.

```
#include <stdio.h>
int main(void)
{
    int i, j;                                   // 반복 제어 변수
    for (i = 0; i < 5; i++)                     // i는 0부터 4까지 다섯 번 반복, 행의 수
    {
        for (j = 0; j < 5; j++)                 // j는 0부터 4까지 다섯 번 반복, 열의 수
        {
            if ((i == j) || (i + j == 4))       // 대각선의 위치가 되었을 때
                printf("*");                    // 별 출력
            else                                // 그 외의 위치는
                printf(" ");                    // 빈칸 출력
        }
        printf("\n");                           // 한 행이 끝나면 줄을 바꾼다.
    }
    return 0;
}
```

이 문제를 풀려면 이중 for문이 화면에 어떤 모양으로 출력하는지를 이해해야 합니다. 바깥쪽 for문은 행(row)을 바꾸고 안쪽 for문은 열(column)을 바꾸면서 5행 5열의 화면에 출력합니다. 단, 빈칸과 별 중에 어떤 것을 출력할지는 행과 열을 결정하는 i와 j의 값에 따라 결정됩니다. 오른쪽 아래로 이어지는 대각선은 i와 j의 값이 같으며 왼쪽 아래로 이어지는 대각선은 i와 j를 더한 값이 4가 되므로 이 조건을 만족할 때 별을 출력하도록 if문을 작성합니다.

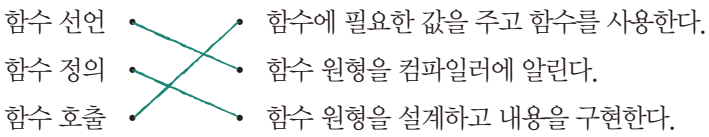
07-1 함수의 작성과 사용

1. 답 ① double ② double a, double b

① 두 실수의 평균도 실수이므로 반환값의 형태는 double.

② 호출할 때 2개의 실수가 전달되므로 double형 매개변수 선언.

2.



3.

```
#include <stdio.h>
double centi_to_meter(int cm);      // 함수 선언
int main(void)
{
    double res;                     // 함수의 반환값을 저장할 변수
    res = centi_to_meter(187);      // 함수 호출, 반환값을 res에 저장
    printf("%.2lfm\n", res);        // 반환된 res의 값 출력

    return 0;
}

double centi_to_meter(int cm)       // 함수 정의 시작
{
    double meter;                   // 필요한 변수 선언
    meter = cm / 100.0;             // 매개변수 cm의 값을 m단위로 환산
    return meter;                   // 환산된 값 반환
}
```

07-2 여러 가지 함수 유형

1. 답 ④

① void func(int, double); ---- func(1.5, 10); → 매개변수의 순서가 int, double 이므로 호출할 때 정수, 실수가 되어야 합니다.

② int func(void); ---- func(void); → 함수를 호출할 때는 void를 사용하지 않습니다.

③ void func(void); ---- func() + 10; → 반환값이 없는 함수는 호출문을 수식의 일

부로 사용할 수 없습니다.

④ `int func(double);` ---- `printf("%d", func(3.4));` → func 함수가 반환하는 값이 printf 함수에 의해 출력됩니다.

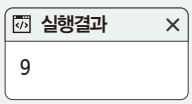
2.

```
#include <stdio.h>
void sum(int n);                    // 함수 선언
int main(void)
{
    sum(10);                        // 1부터 10까지의 합 출력
    sum(100);                       // 1부터 100까지의 합 출력
    return 0;
}
void sum(int n)                     // 매개변수에 합을 구할 마지막 값을 받는다.
{
    int i, tot = 0;                 // 반복 회수를 세는 변수와 합을 누적할 변수 정의
    for(i = 1; i <= n; i++)         // 1부터 매개변수 n까지 i 증가
    {
        tot += i;                   // i를 반복해 tot에 누적한다.
    }
    printf("1부터 %d까지의 합은 %d입니다.\n", n, tot);   // n과 tot 출력
}
```

3.

```
int func(int n);   // 2*n*n + 3*n을 계산한 후에 그 절대값을 구하는 함수
int poly(int n);   // 2*n*n + 3*n을 계산해 반환하는 함수
```

답



func 함수는 poly 함수가 반환한 2 * n * n + 3 * n 값의 절대값을 구해 반환합니다.

08-1 배열의 선언과 사용

1. 답 ① `int ary[5];` ② `double ary[10];` ③ `int ary[3]` ④ `char ary[5];`

2. 답 `int ary[6] = { 1, 2, 3 }`

배열 요소 개수가 6개인 int형 배열을 선언하고 처음 3개만 초기화하면 나머지 배열 요소는 0으로 자동 초기화됩니다.

3.

```c
#include <stdio.h>

int main(void)
{
    int A[3] = { 1, 2, 3 };        // 초기화된 A 배열
    int B[10];                     // 초기화되지 않은 B 배열
    int i;

    for (i = 0; i < 10; i++)       // B 배열을 채우기 위해 B 배열 요소의 개수만큼 반복
    {
        B[i] = A[i % 3];           // A 배열 첨자가 0 ~ 2를 갖도록 나머지 연산자 사용
    }

    for (i = 0; i < 10; i++)
    {
        printf("%5d", B[i]);       // B 배열 출력
    }
    return 0;
}
```

실행결과

```
    1    2    3    1    2    3    1    2    3    1
```

08-2 문자를 저장하는 배열

1. 답 ① ○ ② ○ ③ × ④ ×

① 남는 배열 요소에는 자동으로 0이 채워집니다.

② 문자열 끝의 널 문자를 포함해 9개의 배열 요소를 할당합니다.

③ 널 문자를 저장할 공간이 없습니다.

④ 배열의 크기가 작고, 중괄호 없이 하나의 문자열만 초기화할 수 있습니다.

2. 답 ④

3. 답 ① temp ② str1 ③ str1 ④ str2 ⑤ str2 ⑥ temp

① temp ② str1 → temp 배열에 str1 배열의 문자열 대입

③ str1 ④ str2 → str1 배열에 str2 배열의 문자열 대입

⑤ str2 ⑥ temp → str2 배열에 temp 배열의 문자열 대입

09-1 포인터의 기본 개념

1. 답 ① `char *p;` ② `int *p;` ③ `double *p;`

2. 답

수식	&ch	&in	&db	*&ch	*&in	*&db
결괏값	100	101	105	'A'	10	3.4

&ch, &in, &db는 각 변수가 할당된 메모리의 시작 주소 값입니다.

*&ch는 &ch의 연산 결과인 주소 100에 다시 간접 참조 연산을 수행해 주소 위치의 변수 ch의 값이 됩니다. 보통 & 연산으로 구한 주소는 포인터에 저장해 쓰지만, 주소 값을 바로 * 연산에 사용하는 것도 가능합니다.

3.

```
int a = 10;                 // a를 10으로 초기화
int *p = &a;                // a의 주소를 p에 저장해 p가 a를 가리키도록 초기화
*p = 20;                    // p가 가리키는 변수(a)에 20 대입
printf ("%d", a) ;          // 20으로 바뀐 a값 출력
```

답

실행결과

```
20
```

09-2 포인터 완전 정복을 위한 포인터 이해하기

1. 답 상수 ②, ⑤ / 변수 ①, ③, ④

&a는 변수 a의 시작 주소 값이므로 주소 상수

p는 포인터 변수, *p는 p가 가리키는 변수 a

2. 답 ④

pc와 pd는 포이터이므로 크기가 4바이트로 같습니다.

*pc는 pc가 가리키는 변수의 크기이므로 1바이트

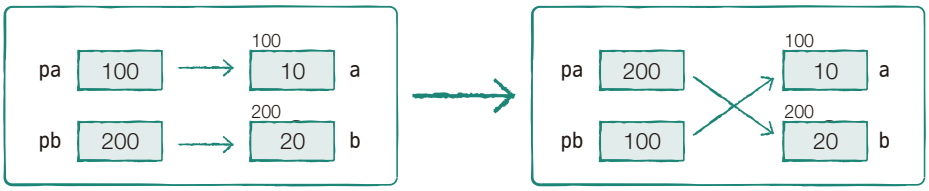
*pd는 pd가 가리키는 변수의 크기이므로 8바이트

3. 답

실행결과

```
20, 10
```

변수 a, b를 가리키는 포인터 pa, pb의 값을 바꾸어 pa가 b를 가리키고 pb가 a를 가리키도록 합니다. 따라서 *pa, *pb는 각각 b와 a의 값을 출력합니다. pa, pb를 교환할 때 사용하는 변수는 pa, pb와 같은 형태의 포인터를 사용합니다.



10-1 배열과 포인터의 관계

1. 답 ① 100 ② 3.5 ③ 116 ④ 0.5 ⑤ 7.4 ⑥ 2

① 배열명은 첫 번째 배열 요소 ary[0]의 주소

② 포인터 연산식 *(ary + 1)은 배열 요소 표현식 ary[1]과 같습니다.

③ pa가 100이고 가리키는 자료형이 double이므로 100 + (2 * sizeof(double))

④ pa[3]는 *(pa + 3)과 같습니다.

⑤ pb는 ary + 2로 초기화되었으므로 *pb는 세 번째 배열 요소가 됩니다.

⑥ (116 – 100) / sizeof(double)

2. 답 ① × ② × ③ ○ ④ ○ ⑤ × ⑥ ○

① 배열 요소의 수가 5개이므로 첨자는 0~4만 사용해야 합니다.

② ary는 배열명으로 주소 상수 100이므로 증가 연산자를 사용할 수 없습니다.

③ *ary는 첫 번째 배열 요소이므로 ++(*ary)는 그 값을 증가시킵니다.

④ 포인터 연산식으로 바꾸면 *(pb – 2)가 되고 pb의 값 116에서 가리키는 자료형의 크기(8)를 곱해서 빼 주면 100이므로 결국 첫 번째 배열 요소의 주소가 됩니다. 여기에 간접 참조 연산을 수행하므로 결국 첫 번째 배열 요소를 사용합니다.

⑤ 초기화된 pb는 세 번째 배열 요소를 가리키므로 pb + 3은 배열의 할당 영역을 벗어납니다.

⑥ ++pa는 전위형이므로 먼저 pa가 두 번째 배열 요소를 가리키도록 하고 이어서 간접참조 연산으로 두 번째 배열 요소를 사용합니다.

3. 답 pb++; 또는 ++pb;

pb가 가리키는 배열 요소를 출력하고 다음 배열 요소로 이동해야 하므로 증가 연산자를 사용합니다.

10-2 배열을 처리하는 함수

1. 답 ③

매개변수가 포인터이므로 주소를 인수로 주는 호출이면 모두 가능합니다. ①, ②의 배열명 ary는 주소이므로 호출이 가능하고 ④의 ary + 2는 세 번째 배열 요소의 주소를 주고 호출합니다. 이 경우 배열의 중간부터 데이터를 처리할 때 사용합니다. ③의 ary는 double형 배열이므로(int *)형 포인터에 대입하면 안 됩니다.

2.

```
void print_month(int *mp)                          // 배열명을 저장할 포인터
{
    int i;                                         // 반복 제어 변수
    for (i = 0; i < 12; i++)                       // 출력 열두 번 반복
    {
        printf("%5d", mp[i]);                      // 각 달의 일수 출력
        if((i + 1) % 5 == 0) printf("\n");         // 출력 개수가 5의 배수면 줄 바꿈
    }
}
```

11-1 아스키 코드 값과 문자 입출력 함수

1. 답 ①, ③, ④, ⑥

① 'a' → 소문자 'a'의 아스키 코드 값은 97

② 'A' – 32 → 대문자 'A'의 아스키 코드 값은 65, 'A' - 32는 33

③ 'b' - 1 → 소문자 'b'의 아스키 코드 값은 98, 'b' – 1의 값은 97

④ 'A' + ('b' - 'B') → 대문자 'A'의 아스키 코드 값 65, 소문자와 대문자의 차는 32

⑤ 97 – '0' → 문자 '0'의 아스키 코드 값은 48, 97–48=49

2.

```
#include <stdio.h>
int main(void)
{
    char ch;

    printf("문자 입력 : ");
    scanf("%c", &ch);                                   // 문자 입력
    printf("%c문자의 아스키 코드 값 : %d\n", ch, ch);    // %d로 아스키 코드 값 출력
    return 0;
}
```

11-2 버퍼를 사용하는 입력 함수

1. 답 ②

① `fflush (stdout);` → 출력 버퍼의 내용을 지울 때 사용

③ `fgetc(stdin);` → 버퍼에서 하나의 문자만 가져와 반환

④ `scanf("%c", &ch)` ; → 버퍼에서 하나의 문자만 가져와 ch에 저장

2. 답 ②, ⑤, ⑥

② `ch = scanf("%c");` → scanf 함수의 반환값은 입력에 성공한 데이터의 개수

⑤ `getchar(&ch);` → getchar 함수는 인수가 없음, `getchar()`와 같이 사용

⑥ `scanf("%c", &num)` → 문자를 입력할 때는 char형 변수 사용

3. 답 ① `'\n'`, ② `(ch >= 'a') && (ch <= 'z')`

① 입력 버퍼의 끝에 있는 개행 문자 전까지 입력

② 소문자의 아스키 코드 값 범위를 검사

12-1 문자열과 포인터

1. 답 `char *ps`

char 포인터와 char형 배열 모두 문자열로 초기화할 수 있으나 ps += 5와 같이 ps 값을 바꾸는 일은 포인터만 할 수 있습니다. 배열명은 상수이므로 값을 바꿀 수 없습니다.

2. 답 ② `printf("%s", str[0]);`

`str[0]`는 char형 배열의 첫 번째 요소며 `'a'`가 저장되어 있습니다. 따라서 `printf` 함수의 인수로 소문자 a의 아스키 코드 값 97이 전달되며 `printf` 함수는 97번지부터 널 문자가 나올 때까지 문자열을 출력합니다. 메모리 97번지가 어떤 용도로 사용될지 알 수 없으므로 실행결과는 예상할 수 없습니다.

3. 답 `fgetc(stdin);`

`scanf` 함수로 좋아하는 동물 이름을 입력한 후에는 버퍼에 남아 있는 개행 문자가 `fgets` 함수의 입력으로 사용되므로 좋아하는 이유를 입력하지 못합니다. 따라서 `scanf("%s", ani);` 문장 다음 행에 `fgetc(stdin);` 문장을 추가해 버퍼에 남아 있는 개행 문자를 지웁니다.

12-2 문자열 연산 함수

1. 답 ② ⑤ ⑥

② `strcat(str, ps);` → str 배열에 붙여 넣을 공간이 부족합니다.

⑤ `strcat("cute", str);` → "cute" 뒤에 str을 붙여 넣을 공간이 없습니다.

⑥ `strncpy(ps, str, strlen(str));` → ps가 연결하고 있는 문자열 상수는 바꿔서는 안 됩니다.

2.

```
strcpy(str, "wine");                    // str 배열에 wine 복사
strcat(str, "apple");                   // wine 뒤에 apple을 붙임
strncpy(str, "pear", 1);                // 문자열 pear에서 첫 번째 문자만 str에 복사
// 복사한 후에 널 문자를 붙이지 않으므로 wineapple이 pineapple로 바뀜
printf("%s, %d\n", str, strlen(str));   // str 배열의 문자열과 그 길이를 구해 출력
```

실행결과

```
pineapple, 9
```

3.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
```

```
    char str[80];                                   // 문자열을 입력할 배열
    char res_str[80];                               // 생략 문자열을 저장할 배열
    char *star = "**********";                      // 생략 부분을 채울 문자열
    int len;                                        // 입력 문자열의 길이 보관
    printf("단어 입력 : ");
    scanf("%s", str);
    len = strlen(str);                              // 입력한 단어의 길이 계산
    if (len <= 5)                                   // 길이가 5 이하이면 그대로 복사
    {
        strcpy(res_str, str);
    }
    else                                            // 5보다 크면
    {
        strncpy(res_str, str, 5);                   // 일단 다섯 문자만 복사
        res_str[5] = '\0';                          // 마지막에 널 문자 저장
        strncat(res_str, star, len - 5);            // 문자열의 길이만큼 별로 채움
    }
    printf("입력한 단어 : %s, 생략한 단어 : %s\n", str, res_str);
    return 0;
}
```

13-1 변수 사용 영역

1. 답 ③ ④ ⑤ ⑥

③ 전역 변수의 사용 범위가 더 큽니다.

④ auto를 사용한 지역 변수는 자동 초기화되지 않습니다.

⑤ 하나의 함수 안이라도 블록을 새로 열면 같은 이름의 변수를 선언할 수 있습니다.

⑥ 지역 변수에 우선권이 있습니다.

2. 답

실행결과

```
30
```

a는 전역 변수이므로 main 함수와 func 함수에서 모두 사용할 수 있습니다. 따라서 func 함수에서 마지막으로 대입한 30이 출력됩니다.

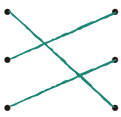
3. 답 static

func 함수가 반환하는 값이 1부터 10까지 계속 증가하려면 함수 안에 선언한 a값이 계속 유지되도록 정적 지역 변수로 선언해야 합니다.

13-2 함수의 데이터 공유 방법

1. 답

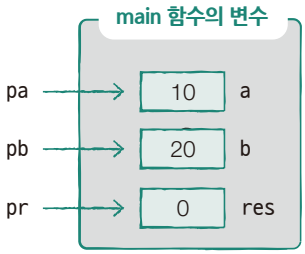
```
void swap(int *pa, int *pb);
double avg(int a, int b);
char *get_str(void);
```



값을 복사해서 전달하는 방법
주소를 반환하는 함수
주소를 전달하는 방법

2. 답 지역 변수인 n의 주소를 반환하고 있습니다. 3행을 static int n;과 같이 작성해야 합니다.

3. 답 *pr = *pa + *pb;



add_by_pointer 함수는 반환값을 사용하지 않고 포인터인 매개변수를 통해 인수를 받고 결과값도 매개변수 pr를 통해 호출한 함수의 변수 res에 직접 저장합니다.

14-1 다차원 배열

1. 답 ① int stock[25][200]; → 재고량은 정수이므로 int형 배열 선언

② double sight[50][2]; → 시력은 소수점이 있으므로 double형 배열 선언

③ char word[15000][46]; → 마지막 널문자까지 포함해 각 행의 길이는 46열로 선언

2. 답 ② ④ ⑤

② 초기화를 하더라도 열의 수는 생략할 수 없습니다.

④ "banana"는 널 문자까지 포함해 7자이므로 열의 길이를 7로 선언해야 합니다.

⑤ 열의 수가 생략되었습니다.

3. 답 ① `i == j`　　② `(i == j) || (i == (4 - j))`

14-2 포인터 배열

1. 답 `char *pary[5] = { "apple", "pear", "peach", "banana", "melon" };`

2. 답 hope

3. 답 ④

pary는 배열 요소가 4개인 배열이므로 첨자는 0부터 3까지만 사용해야 합니다.

15-1 이중 포인터와 배열 포인터

1. 4.5 → ppg가 가리키는 포인터가 가리키는 변수의 값, 즉 grade의 값
300 → ppg의 주소
100 → pg의 주소가 가리키는 변수의 값, 결국 pg의 값
100 → ppg가 가리키는 변수의 값, 결국 pg의 값
200 → ppg가 가리키는 변수의 주소, 결국 pg의 주소

답

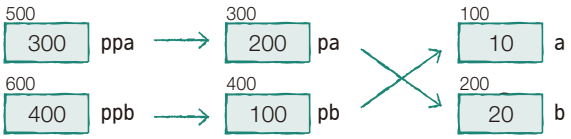
실행결과

```
4.5
300
100
100
200
```

2. 답

실행결과1

```
a:10, b:20
*pa:20, *pb:10
```



이중 포인터로 pa, pb의 값을 바꾼 것이며 a, b의 값은 변함이 없습니다. 그러나 pa, pb가 가리키는 대상이 바뀌었으므로 pa, pb를 통해서 a, b의 값을 서로 바꿔 사용할 수 있습니다.

3. 답 ① `char **p;` ② `int (*p)[3];` ③ `int *p;` ④ `char **p;` ⑤ `int (*p)[2][3];`

① 포인터 배열의 배열명은 이중 포인터로 받음

② 2차원 배열의 배열명은 배열 포인터로 받음

③ 부분배열 `ary[0]`는 자체가 1차원 배열임

④ `ps[2]`는 포인터이므로 그 주소인 `&ps[2]`는 이중 포인터로 받음

⑤ 2행 3열의 2차원 배열 전체를 가리키는 배열 포인터

15-2 함수 포인터와 void 포인터

1. ① `double (*fpa)(int, int);` ② `void (*fpb)(char *);` ③ `int *(*fpc)(int);`

2. 답 `printf("%d", ((int *)vp)[2]);`

형 변환 연산자는 배열 연산자보다 우선순위가 낮으므로 형 변환이 먼저 수행되도록 괄호를 사용해야 합니다. 배열 연산자는 배열 요소를 쓸 때 사용하는 대괄호이며 배열명에 첨자를 더하고 간접참조 연산자를 사용하는 포인터 표현과 같은 연산입니다.

```
ary[2]   ->   *(ary + 2)
```

3. 답

실행결과

```
6
```

`pary`는 함수 포인터 3개를 요소로 갖는 배열로 `pary[0]`, `pary[1]`, `pary[2]`는 각각 `add`, `sub`, `mul`을 가리키는 함수 포인터입니다. 따라서 2와 1의 합, 차, 곱을 `res`에 누적해 출력합니다.

16-1 동적 할당 함수

1. 답 ① `(double *) malloc(sizeofdouble));`

② `(int *) malloc(10 * sizeof(int));`

③ `(char *) malloc(80 * sizeof(char));`

2. 답 ① `max == 0` 또는 `max == NULL` ② `free(max);`

3. 답 `pa`

`sizeof(ary)`는 배열 전체의 크기이므로 `pa`는 20바이트를 할당합니다. 이어서 `pa[3]`는 `pa`를 배열처럼 사용할 때 네 번째 요소의 값이므로 `pb`는 `4 * sizeof(int)` 크기인 16바이트를 할당합

니다. realloc 함수는 첫 번째 인수가 NULL이면 두 번째 인수의 바이트 크기만큼 동적 할당하며 pa[4]는 5이므로 pc는 5바이트를 할당합니다.

16-2 동적 할당 저장 공간의 활용

1.

```
int **matrix = (int **) malloc(4 * sizeof(int *));
// 포인터 배열로 사용할 공간의 동적 할당
for (i = 0; i < 4; i++)
{
    matrix[i] = (int *) malloc(5 * sizeof(int));
    // 각 행을 동적 할당해 포인터 배열에 연결
}
```

2.

```
for (i = 0; i < 4; i++)     // 각 행의 동적 할당 영역 반환
{
    free(matrix[i]);
}
free(matrix);               // 포인터 배열로 사용한 동적 할당 영역 반환
```

3. 답 ① int argc, char **argv ② argc - 1 ③ argv[i + 1]

17-1 구조체

1.

```
struct book
{
    char title[30];
    char author[20];
    int page;
    int price;
};
```

2.

```
#include <stdio.h>
struct cracker      // 구조체 선언
{
    int price;      // 가격을 저장할 멤버
    int calories;   // 열량을 저장할 멤버
};
int main(void)
{
    struct cracker basasak;                                    // 구조체 변수 선언
    printf("바사삭의 가격과 열량을 입력하세요 : ");
    scanf("%d%d", &basasak.price, &basasak.calories);          // 멤버에 값 입력
    printf("바사삭의 가격 : %d원\n", basasak.price);             // 입력된 값 출력
    printf("바사삭의 열량 : %dkcal\n", basasak.calories);
    return 0;
}
```

3. 답 ① ② ③ ④

① strcpy 함수는 첫 번째 인수로 받은 주소에 문자열을 복사하므로 event에 복사받을 공간을 먼저 확보한 후에 복사해야 합니다.

```
a.event = (char *)malloc(80);           // 80바이트 저장 공간을 확보한 후에
strcpy(a.event, "figure skating");      // 동적 할당한 공간에 문자열 복사
```

② name은 배열명으로 주소 상수이므로 대입 연산자 왼쪽에 올 수 없습니다.

③ num은 struct profile의 멤버이므로 a.player.num과 같이 멤버에 접근해야 합니다.

④ skill도 포인터이므로 입력한 문자열을 저장할 공간을 먼저 확보한 후에 입력해야 합니다.

17-2 구조체 활용, 공용체, 열거형

1.

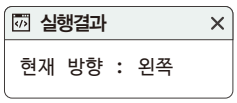
```
printf("이름 : %s", mp->name);
printf("나이 : %d", mp->age);
printf("성별 : %c", mp->gender);
printf("키 : %.1lf", mp->height);
```

2. 답 ① Train 또는 struct train ② tail = tail->next;

```
typedef struct train Train;   // struct train을 Train으로 형 재정의
struct train
{
    int seats;
    Train *next;              // 먼저 struct train을 재정의했으므로 Train 사용 가능
};
```

tail 포인터가 항상 연결 리스트의 마지막 위치를 기억하도록 새로운 구조체 변수를 연결한 후에는 tail의 값을 마지막 위치로 바꿉니다.

3. 답




```
typedef enum { CYAN, MAGENTA, YELLOW = 5, BLACK } COLOR;
typedef enum { UP, DOWN, LEFT, RIGHT } ARROW;
```

열거형 선언과 동시에 COLOR와 ARROW형으로 재정의합니다. COLOR의 열거 멤버 CYAN은 0, MAGENTA는 1, YELLOW는 5, BLACK은 6의 값을 가지며 ARROW 멤버의 값은 0부터 차례로 1씩 증가합니다.

```
for (c = CYAN; c <= BLACK; c++)      // 0부터 6까지 일곱 번 반복
{
    direction++;                     // direction을 1 증가
    direction = direction % 4;       // direction은 1, 2, 3, 0, 1, 2, 3...으로 순환
    if (c == my_color) break;        // my_color가 YELLOW이므로 5에서 반복 종료
}
```

결국 c는 0부터 5까지 여섯 번 반복하므로 direction의 최종값은 2, 방향은 왼쪽이 됩니다.

18-1 파일 개방과 입출력

1. 답 ②, ⑤

② 파일 포인터는 스트림 파일에 있는 FILE 구조체 변수의 주소를 저장합니다.

⑤ EOF는 파일의 끝을 의미하는 단어로 그 값은 -1입니다.

2. 답 ②, ③, ④

② 입출력 함수가 같은 파일 포인터를 사용하면 스트림 파일의 버퍼를 공유합니다.

③ 프로그램에서 필요하다면 여러 파일을 동시에 개방할 수 있습니다.

④ 파일의 끝을 표시하는 별도의 값이 파일에 저장되는 것은 아닙니다.

3. 답 ① FILE ② "r" ③ 0 또는 NULL

18-2 다양한 파일 입출력 함수

1. 답 ③ → 나머지 함수의 반환값 형태는 모두 int형이며 fgets 함수는 char *형입니다.

2. 답 ②

```
char str[5];
char ch;
int i = 0;

ch = fgetc(stdin);          // 표준 입력 스트림 파일을 사용하므로 키보드로 문자 입력

while ((i < (sizeof(str) - 1)) && (ch != '\n'))  // 배열에 여유가 있고 개행문자가 아니면
{
    str[i++] = ch;          // 입력한 문자를 배열에 저장
    ch = fgetc(stdin);      // 다음 문자 입력
}

if (i < (sizeof(str) - 1)) str[i++] = ch;        // 조건이 참이면 개행문자를 배열에 저장
str[i] = '\0';                                   // 마지막에 널 문자 저장
```

키보드로 입력한 한 줄의 데이터를 개행문자까지 배열에 저장하므로 fgets 함수의 기능을 구현한 코드입니다. 단, 파일 포인터는 stdin을 사용합니다.

3. 답

실행결과
```
empty
```

처음 fgetc 함수가 호출될 때 파일의 데이터가 작으므로 모든 데이터가 버퍼로 입력됩니다. 이어서 fgetc 함수를 반복 호출해 버퍼의 모든 내용을 지우므로 이후 호출되는 fgets 함수는 입력할 데이터가 없습니다. 결국 fgets 함수는 널 포인터를 반환하며 str 배열의 초깃값 empty가 그대로 출력됩니다.

19-1 전처리 지시자

1. 답 ④

#ifdef, #ifndef는 조건식으로 매크로명만 사용하며 매크로명의 정의 여부를 확인합니다.

2. 답

실행결과
```
35
```

매크로 함수의 부작용을 확인합니다. 매크로 함수의 호출문장 SUM(20, 5) * 3은 전처리 과정에서 20 + 5 * 3으로 치환되고 컴파일러는 5 * 3을 먼저 계산한 후에 20과 더합니다. 따라서 최종 결괏값은 35가 됩니다.

3.

```
#ifndef DEBUG               // DEBUG 매크로명이 정의되지 않았다면
    flag = 0;
#elif LEVEL == 1            // DEBUG 매크로가 정의되고 LEVEL 매크로가 1이면
    flag = 1;
#elif defined(MAX_LEVEL) && (LEVEL == 2)    // MAX_LEVEL이 정의되고 LEVEL이 2면
    flag = 2;
#else                       // 그 이외의 모든 경우
    flag = 3;
#endif                      // 조건부 컴파일의 끝
```

답

실행결과
```
3
```

19-2 분할 컴파일

1.

```
// main.c
#include <stdio.h>
extern void set_key(int);        // sub.c 파일에 있는 함수의 선언 extern은 생략 가능
extern int get_key(void);        // sub.c 파일에 있는 함수의 선언 extern은 생략 가능
```

```c
int key;            // 전역 변수 선언, sub.c 파일의 key는 static이므로 중복되지 않는다.
int main(void)
{
    int res;
    set_key(10);                // sub.c의 static key에 인수 10 저장
    key = get_key();            // sub.c의 key값을 main.c의 전역 변수 key에 저장
    set_key(20);                // sub.c의 static 전역 변수 key의 값을 20으로 바꿈
    res = key + get_key();      // main.c의 전역 변수 key와 sub.c의 static key의 합
    printf("%d", res);
    return 0;
}
```

```c
// sub.c
static int key;             // 정적 전역 변수로 main.c에서 extern 선언으로 공유할 수 없음
void set_key(int val)       // 정적 전역 변수 key의 값을 설정하는 함수
{
    key = val;
}
int get_key(void)           // 정적 전역 변수 key의 값을 다른 파일에서 사용할 수 있도록 반환
{
    return key;
}
```

답

실행결과

```
30
```

정적 전역 변수로 변수의 사용을 하나의 파일로 제한하면 다른 파일에서는 함수를 통해서만 사용할 수 있으므로 허용되지 않은 접근을 막고 데이터를 보호할 수 있습니다.

2. 답 ④ static 전역 변수의 사용 범위는 파일로 제한되므로 다른 파일에서 공유할 수 없습니다.

3. 답 ③ 초기화한 전역 변수의 선언은 헤더 파일이 여러 파일에 인클루드되었을 때 전역 변수의 중복 문제가 발생합니다.