2023/05/08

APPDONG
C언어 멘토링

4주차
멘토 : 김민수

Contents

1차원 배열 4-1 : 개수 세기

문제

총 N개의 정수가 주어졌을 때, 정수 v가 몇 개인지 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수 N(1 ≤ N ≤ 100)이 주어진다. 둘째 줄에는 정수가 공백으로 구분되어져있다. 셋째 줄에는 찾으려고 하는 정수 v가 주어진다. 입력으로 주어지는 정수와 v는 -100보다 크거나 같으며, 100보다 작거나 같다.

출력

첫째 줄에 입력으로 주어진 N개의 정수 중에 v가 몇 개인지 출력한다.

예제 입력 1 복사

```
11
1 4 1 2 4 2 4 2 3 4 4
2
```

예제 출력 1 복사

```
3
```

예제 입력 2 복사

```
11
1 4 1 2 4 2 4 2 3 4 4
5
```

예제 출력 2 복사

```
0
```

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int N;  // 정수 N (1 <= N <= 100)
    int n[100]; // n[] : 입력되는 정수를 저장하는 배열
    int v, i, cnt = 0;  // v : 찾으려고 하는 정수, cnt는 v의 개수

    scanf("%d", &N);    // 정수 N 입력

    for (i = 0; i < N; i++) {
        scanf(" %d", &n[i]);    // 정수 N개 입력
    }

    scanf(" %d", &v);   // 정수 v 입력

```

문제에서 언급된 변수들 모두 선언

입력

```
18      for (i = 0; i < N; i++) {
19          if (n[i] == v) {
20              cnt++;
21          }
22      }
23
24      printf("%d", cnt);
25
26      return 0;
27  }
```

입력으로 주어지는 정수가 v와 같으면
찾은 개수(cnt) +1

1차원 배열 4-2 : X보다 작은 수

문제

정수 N개로 이루어진 수열 A와 정수 X가 주어진다. 이때, A에서 X보다 작은 수를 모두 출력하는 프로그램을 작성하시오.

입력

첫째 줄에 N과 X가 주어진다. (1 ≤ N, X ≤ 10,000)

둘째 줄에 수열 A를 이루는 정수 N개가 주어진다. 주어지는 정수는 모두 1보다 크거나 같고, 10,000보다 작거나 같은 정수이다.

출력

X보다 작은 수를 입력받은 순서대로 공백으로 구분해 출력한다. X보다 작은 수는 적어도 하나 존재한다.

예제 입력 1 복사

```
10 5
1 10 4 9 2 3 8 5 7 6
```

예제 출력 1 복사

```
1 4 2 3
```

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int N, X;    // (1 <= N <= 10000);
    int A[10000] = { 0, }; // 수열 A[] : N개의 정수 저장
    int i;

    scanf(" %d %d", &N, &X);    // 정수 N, X 입력

```

문제에서 언급된 변수(N, X, A[]) 선언

```
12      for (i = 0; i < N; i++) {
13          scanf(" %d", &A[i]);     // A[] 입력
14      }
15
16      for (i = 0; i < N; i++) {
17          if (A[i] < X) printf("%d ", A[i]);
18      }
19
20      return 0;
21  }
```

반복문으로 A[] 배열 전체를 탐색하면서
X보다 작은 값 출력

1차원 배열 4-3 : 최소, 최대

문제

N개의 정수가 주어진다. 이때, 최솟값과 최댓값을 구하는 프로그램을 작성하시오.

입력

첫째 줄에 정수의 개수 N (1 ≤ N ≤ 1,000,000)이 주어진다. 둘째 줄에는 N개의 정수를 공백으로 구분해서 주어진다. 모든 정수는 -1,000,000보다 크거나 같고, 1,000,000보다 작거나 같은 정수이다.

출력

첫째 줄에 주어진 정수 N개의 최솟값과 최댓값을 공백으로 구분해 출력한다.

예제 입력 1 복사

```
5
20 10 35 30 7
```

예제 출력 1 복사

```
7 35
```

문제 해결 방법 #1 : 배열을 사용해서 해결

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int n[1000000];

int main() {
    int N; // (1 <= N <= 1,000,000)
    int i, max = -1000000, min = 1000000;

    scanf("%d", &N);    // 정수 N 입력

```

입력 받은 정수를 저장할 배열을 "전역변수"로 선언

문제에서 언급된 힌트를 이용해서 변수 선언

전역변수로 선언하는 이유?
일반적으로 함수 내부에 선언하는 변수들은 stack 영역에 저장된다.
하지만 많은 양의 데이터가 들어오게 되면 메모리가 부족하여 실행되지 않는다.
이를 해결하기 위해 보다 많은 양의 데이터를 다룰 수 있는 heap 영역에 저장되는
전역변수로 선언한다.

```
13      for (i = 0; i < N; i++) {
14          scanf(" %d", &n[i]);    // 정수 n[i] 입력 (-1,000,000 <= n <= 1,000,000)
15
16          if (max < n[i]) max = n[i];
17          if (n[i] < min) min = n[i];
18      }
19
20      printf("%d %d", min, max);
21
22      return 0;
23  }
```

처음에 max = -1,000,000 min = 1,000,000 으로
초기화를 했기 때문에
첫번째 반복 수행에서 max = n[0], min = n[0]가 된다.
이후 반복 수행에서 max, min이 최신화가 된다.

문제 해결 방법 #2 : 배열을 사용하지 않고 해결

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

// 배열을 사용하지 않고 풀기

int main() {
    int N; // (1 <= N <= 1,000,000)
    int n, i, max = -1000000, min = 1000000;

    scanf("%d", &N);    // 정수 N 입력

```

N : 입력 받을 정수의 개수
n : 입력 받을 정수

```
13      for (i = 0; i < N; i++) {
14          scanf(" %d", &n);
15
16          if (max < n) max = n;
17          if (n < min) min = n;
18      }
19
20      printf("%d %d", min, max);
21
22      return 0;
23  }
```

n을 입력 받고 나서

바로 max, min과 비교해서 최신화를 한다.

입력되었던 정수 값들을 다시 사용하지 않기 때문에
이처럼 단순한 작업 같은 경우에는 배열을 사용하지 않아도 풀 수 있다.
특히 이 문제의 경우 입력 개수가 1 ~ 1,000,000 사이로 매우 많은데,
1,000,000개의 데이터를 저장하는 배열을 선언 안해도 되기 때문에
메모리를 상당히 절약할 수 있다.

1차원 배열 4-4 : 최댓값

문제

9개의 서로 다른 자연수가 주어질 때, 이들 중 최댓값을 찾고 그 최댓값이 몇 번째 수인지를 구하는 프로그램을 작성하시오.

예를 들어, 서로 다른 9개의 자연수

3, 29, 38, 12, 57, 74, 40, 85, 61

이 주어지면, 이들 중 최댓값은 85이고, 이 값은 8번째 수이다.

입력

첫째 줄부터 아홉 번째 줄까지 한 줄에 하나의 자연수가 주어진다. 주어지는 자연수는 100 보다 작다.

출력

첫째 줄에 최댓값을 출력하고, 둘째 줄에 최댓값이 몇 번째 수인지를 출력한다.

예제 입력 1 복사

```
3
29
38
12
57
74
40
85
61
```

예제 출력 1 복사

```
85
8
```

문제 해결 방법 #1 : 배열을 사용해서 해결

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int n[9], i, max = -1, max_idx = 0;

```

n[] : 입력 받은 자연수를 저장하는 배열
max : 최댓값
max_idx : 최댓값의 위치

```
8       for (i = 0; i < 9; i++) {
9           scanf(" %d", &n[i]);    // 정수 9개 입력
10
11          if (max < n[i]) {
12              max = n[i];
13              max_idx = i;
14          }
15      }
16
17      printf("%d₩n%d", max, max_idx + 1);
18
19      return 0;
20  }
```

max 값 비교 후
max 값과 max_idx 갱신

출력할 때는 max_idx 값을
+1 해서 출력

문제 해결 방법 #2 : 배열을 사용하지 않고 해결

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int n, i, max = -1, max_idx = 0;

```

n : 입력 받을 자연수
max : 최댓값
max_idx : 최댓값의 위치

```
8     for (i = 1; i <= 9; i++) {
9         scanf(" %d", &n)
10
11        if (max < n) {
12            max = n;
13            max_idx = i;
14        }
15    }
16
17    printf("%d₩n%d", max, max_idx);
18
19    return 0;
20}
```

자연수 n 입력 받은 후

바로 max 값과 비교
max 값 위치도 저장

i 범위를 1 ~ 9로 설정한 이유
: 배열을 사용하지 않았기 때문에 index관리를 하지 않아도 되고
출력할 때 바로 위치를 출력할 수 있다.

포인터 기초

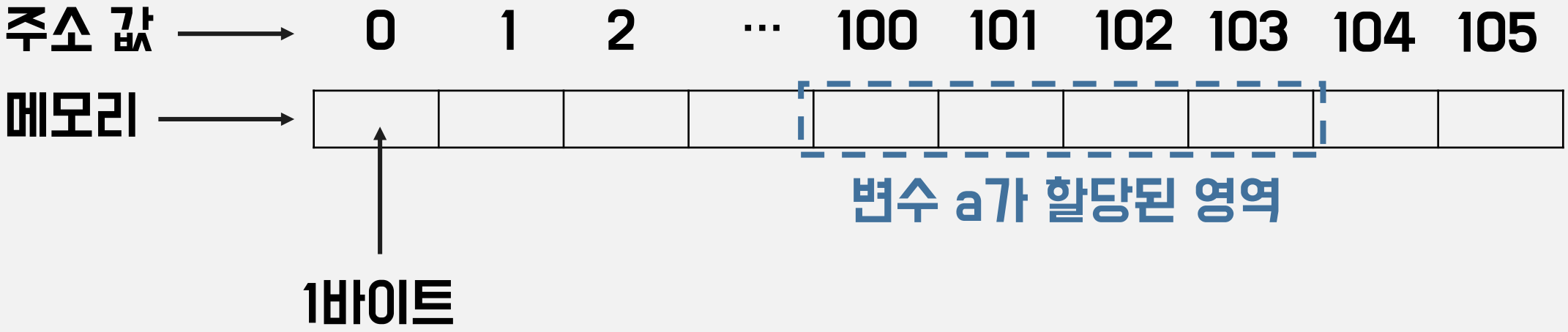
메모리의 주소

데이터를 저장 해놓는 메모리의 위치를 주소 값으로 식별한다

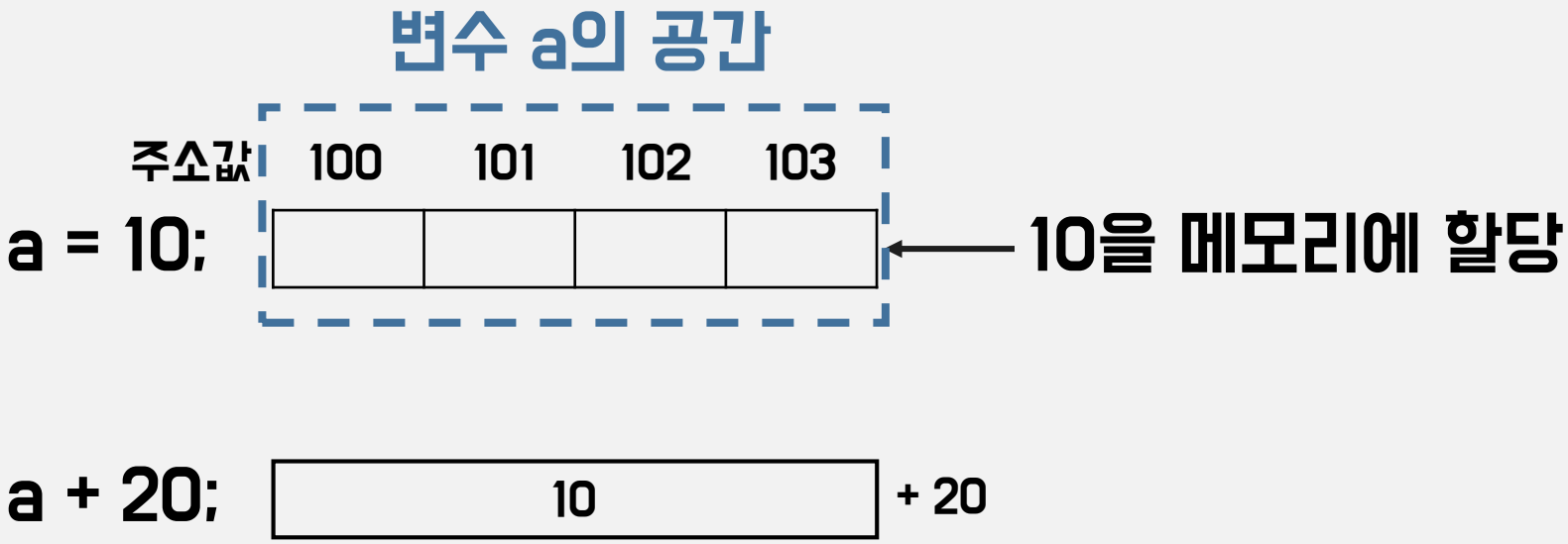
주소 값은 0부터 시작하고 바이트 단위로 1씩 증가하며
2바이트 이상의 크기를 갖는 변수는 여러 개의 주소 값에 걸쳐 할당된다

포인터 기초

int a;



포인터 기초



a = 10; -> 메모리의 100번지에서 103번지까지 4바이트 공간에 10을 저장한다
a + 20; -> 메모리 100번지부터 103번지까지 4바이트에 저장된 값과 20을 더하는 연산을 수행한다

포인터 기초

주소 연산자 &

여기서 말하는 "주소"는 변수가 할당된 메모리 공간의 시작 주소를 의미한다
주소 연산자를 사용해서 주소 값을 알 수 있다

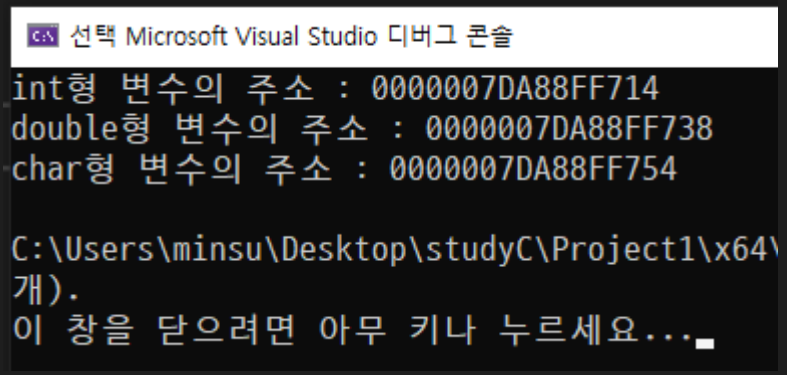
&(피연산자)로 사용해서 피연산자(변수)의 메모리 주소를 반환(return)할 수 있다

주소는 보통 16진수로 표현하기 때문에 주소를 출력할 때는 %p를 사용해서
출력하면 된다

scanf(); 를 사용할 때 변수의 메모리의 주소 값이 인자로 사용된다

포인터 기초

```c
#include <stdio.h>

int main() {
    int a;
    double b;
    char c;

    printf("int형 변수의 주소 : %p₩n", &a);
    printf("double형 변수의 주소 : %p₩n", &b);
    printf("char형 변수의 주소 : %p₩n", &c);

    return 0;
}
```



주소 값을 출력할 때는 %p를 사용한다.

포인터 기초

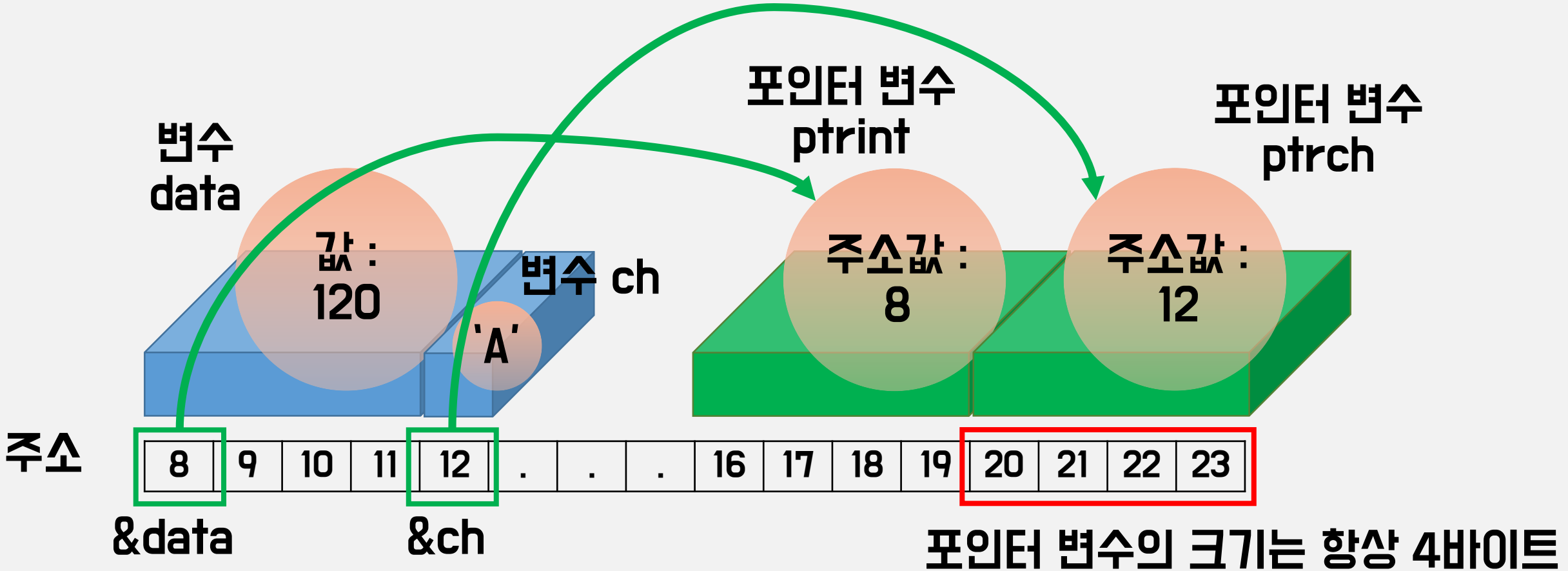
간접 참조 연산자 *

메모리의 주소를 필요할 때마다 계속 주소 연산자로 구하는 것 보다는
한 번 구한 주소를 저장해서 사용하면 편하다

이 때, 포인터를 선언해서 변수의 메모리 주소를 할당하면 된다

포인터 자체는 변수의 메모리 주소 값을 가리키고 있는데
간접 참조 연산자(*)로 변수의 데이터 값에 접근할 수 있다

```c
int data = 120;
char ch = 'A';
int* ptrint = &data;
char* ptrch = &ch;

printf("간접참조 출력 : %d %c", *ptrint, *ptrch);
```

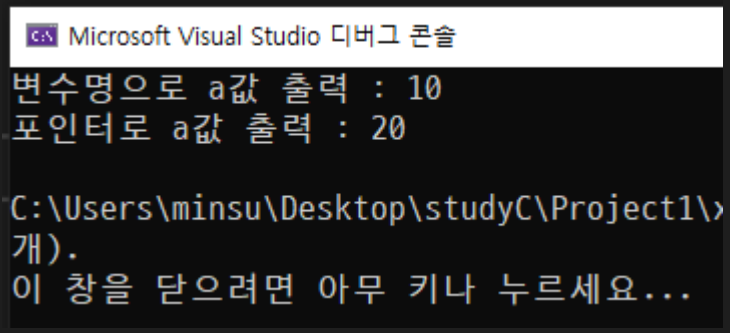


포인터 기초

```c
#include <stdio.h>

int main() {
    int a = 10;
    int* pa;

    pa = &a;

    printf("변수명으로 a값 출력 : %d₩n", a);
    *pa = 20;

    printf("포인터로 a값 출력 : %d₩n", *pa);

    return 0;
}
```

포인터 변수에 a의 주소 할당

간접참조로 a값 변경



포인터 기초

int a;
int *pa;

pa = &a;

값을 입력할 때
scanf("%d", &a); 대신에 scanf("%d", pa); 사용 가능

sizeof 연산자로 포인터 크기를 확인하면 가리키는 자료형과 관계없이
크기가 같다 -> 환경에 따라 다른데 보통 4바이트 또는 8바이트

포인터 변수의 자료형과 가리킬려는 변수의 자료형은 일치해야함
int a; -> int *pa; double b; -> double *pb;

포인터 기초

포인터 상수 표현 (const) 1

```
int a, b;
const int *pa = &a;

*pa = 20;     // 에러

pa = &b;      // 가능
```

pa가 가리키는 변수 a는 pa를 간접참조하여 변경할 수 없다는 것을 의미

단, 다른 변수를 다시 가리키는 것은 가능

포인터 기초

포인터 상수 표현 (const) 2

```
int a, b;
int *const pa = &a;

pa = &b;      // 에러

*pa = 20;     // 가능
```

pa값을 변경할 수 없다는 것을 의미
(다른 변수를 가리킬 수 없음)

단, 가리키는 변수의 값을 간접 참조하여
변경할 수는 있음

포인터 기초

포인터의 덧셈, 뺄셈

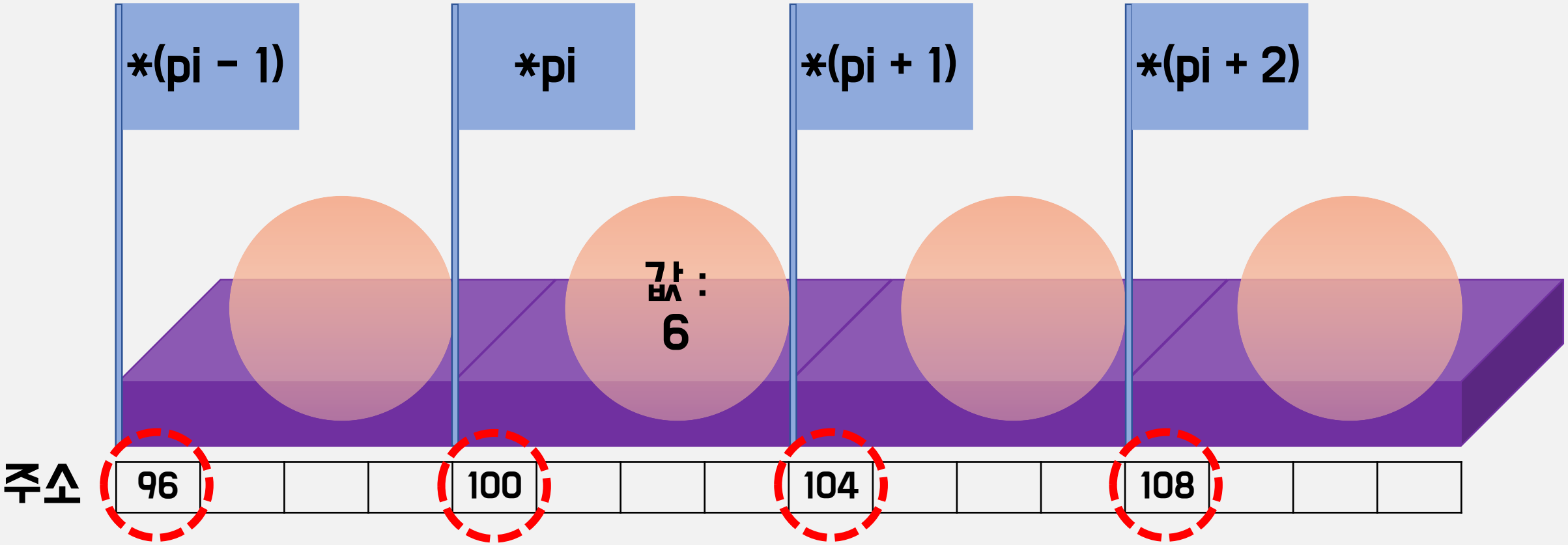
-> 포인터가 가리키는 변수 크기에 비례한 연산
즉, 포인터에 저장된 **주소값**의 연산이다.

```
int data = 6;
int* pi = &data;
```

만약 data의 주소값(&data)이 100일 때,
pi + 1 의 결과는 101이 아닌 104가 된다. (int형이 4바이트이기 때문)

포인터 기초

```
int data = 6;
int* pi = &data;
```

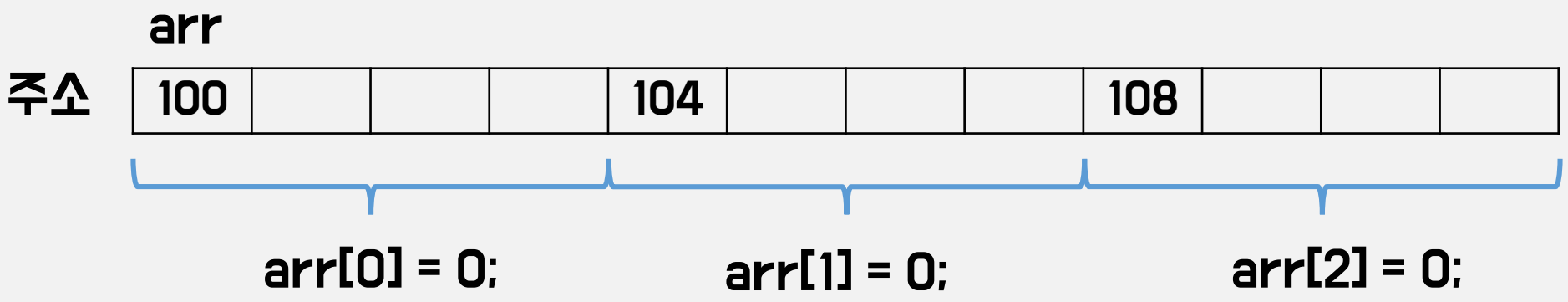


int 형 변수 -> 4바이트

배열과 포인터의 관계

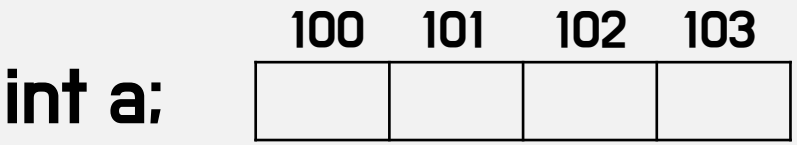
배열은 자료형이 같은 변수를 메모리에 연속으로 할당한다
따라서 각 배열 요소는 일정한 간격으로 주소를 갖게 된다

int arr[3] = {0, };
-> 컴파일러는 배열명(arr)을 첫 번째 배열 요소의 주소로 변경한다



배열과 포인터의 관계

주소 + 정수 -> 주소 + (정수 * 주소를 구한 변수의 크기)

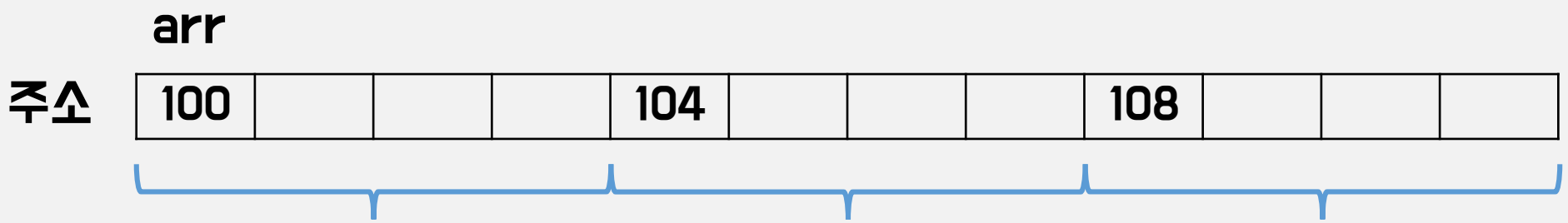


&a : 변수 a의 주소 값 (100)

&a + 1 -> 100 + (1 * sizeof(int)) -> 104

배열은 위 규칙을 잘 이용하여 각 요소에
쉽게 접근할 수 있다

배열과 포인터의 관계



arr[0] = 10;
-> *arr = 10;

arr[1] = 20;
-> *(arr + 1) = 20;

arr[2] = 30;
-> *(arr + 2) = 30;

간접참조연산자 *로 각 배열 원소에 접근해서 값을 변경할 수 있다.

*arr = 10; -> arr[0]에 접근
*(arr + 1) = 20; -> arr[1]에 접근
*(arr + 2) = 30; -> arr[2]에 접근

배열과 포인터의 관계

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int arr[3];

    arr[0] = 10;
    arr[1] = arr[0] + 10;

    printf("arr[2] 값 입력 >> ");
    scanf("%d", &arr[2]);

    for (int i = 0; i < 3; i++) {
        printf("%3d", arr[i]);
    }

    return 0;
}
```

위 코드를 포인터를 사용한 코드로 다시 구현해보자.

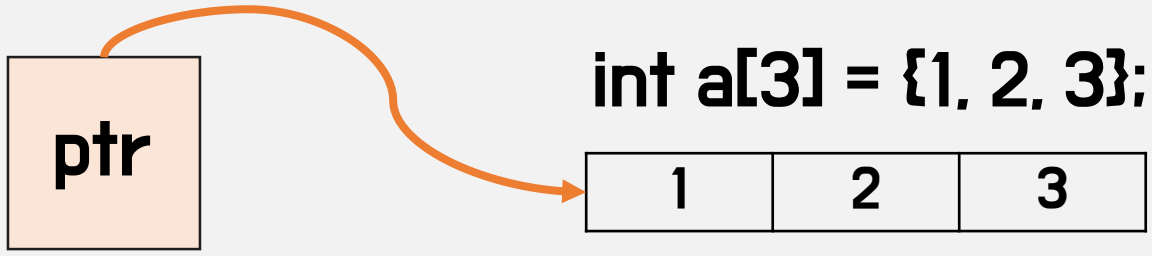
배열과 포인터의 관계

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int arr[3];

    *arr = 10;
    *(arr + 1) = *arr + 10;

    printf("arr[2] 값 입력 >> ");
    scanf("%d", arr + 2);

    for (int i = 0; i < 3; i++) {
        printf("%3d", *(arr + i));
    }

    return 0;
}
```

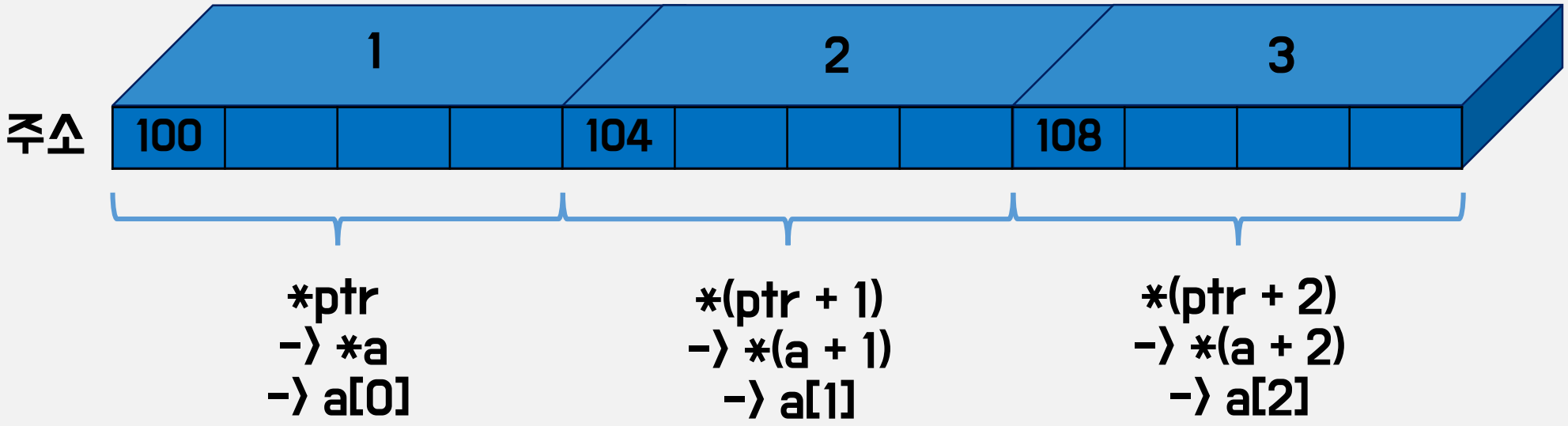
배열과 포인터의 관계

배열명 대신에 포인터 변수를 선언해서 배열 주소를 참조한 후,
배열 원소에 접근할 수 있다 (이 때의 포인터 변수를 "배열 포인터"라고 함.)

int *ptr = a;
-> 배열명을 가리킨다 (첫 번째 주소를 가리킨다)



배열과 포인터의 관계



Q. 배열 이름을 사용해도 되는데 왜 포인터 변수를 또 선언해서 사용을 하는가?

A. 배열 이름은 "포인터 상수"이다. int a[3]; 에서 a 값을 변경할 수 없기 때문에 a++, ++a 같은 a 값을 직접 바꾸는 증감 연산이 불가능하다.
이런 증감 연산을 사용하려면 포인터 변수를 선언해야한다.

배열과 포인터의 관계

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int main() {
    int arr[3];
    int* ptr = arr;

    *ptr = 10;
    *(ptr + 1) = 20;

    printf("arr[2] 값 입력 >> ");
    scanf("%d", ptr + 2);

    for (int i = 0; i < 3; i++) {
        printf("%3d", *ptr++);
    }

    return 0;
}
```

함수

함수를 작성할 때 고려해야할 사항

1. 정의 : 어떤 기능을 구현할 것인지 (여러 번 반복되는 작업인 경우, 복잡한 코드 구조인 경우, 특수한 기능인 경우 등등..)

2. 호출 : 어떤 인자를 통해 함수를 호출할 것인지 (int 변수, char 변수, 포인터 변수 등등..)

3. 선언 : 헤더 파일과 main() 함수 사이에 함수 이름, 인자를 명시적으로 표현

함수

함수 구조

반환 타입 함수이름 (함수인자)
{
	기능 작성;
	return 반환 값; (반환 타입에 따라 return 값 설정)
}

함수

```
1   #define _CRT_SECURE_NO_WARNINGS
2
3   #include <stdio.h>
4
5   int sum_array(int ary[], int size);
6
7   int main() {
8       int arr[5] = { 0, };
9
10      for (int i = 0; i < 5; i++) {
11          scanf("%d", &arr[i]);
12      }
```

함수 선언

함수

```
13
14      printf("평균 점수 : %.4lf₩n", (double)sum_array(arr, 5) / 5);
15
16      return 0;
17  }
18
19  int sum_array(int ary[], int size) {
20      int sum = 0;
21
22      for (int i = 0; i < size; i++) {
23          sum += ary[i];
24      }
25
26      return sum;
27  }
```

함수 호출

함수 정의

함수

* 여러가지 함수 종류 *

1. 매개변수(함수 인자)가 없는 함수
2. 반환 값(리턴 값)이 없는 함수 (=void 타입 함수)
3. 매개변수와 반환 값이 모두 없는 함수
4. 재귀 호출 함수

함수

1. 매개변수가 없는 함수

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

int get_num();

int main() {
    int num;

    do {
        printf("출력 : %d₩n----------₩n", num = get_num());
    } while (num);

    return 0;
}
```

```c

int get_num() {
    int n;

    printf("입력 : ");
    scanf("%d", &n);

    return n;
}
```

함수

2. 반환 값이 없는 함수

```c
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>

void print_char(char ch, int count);

int main() {
    char c;
    int n;

    printf("입력한 문자 여러 번 출력하기₩n");
    printf("문자 입력 >> ");
    scanf("%c", &c);
    printf("숫자 입력 >> ");
    scanf("%d", &n);

    print_char(c, n);

    return 0;
}
```

```c

void print_char(char ch, int count) {
    for (int i = 0; i < count; i++) {
        printf("%c", ch);
    }
}
```

함수

3. 매개변수와 반환 값이 모두 없는 함수

```c
#include <stdio.h>

void print_line();

int main() {
    print_line();
    printf("학번     이름     전공          학점₩n");
    print_line();
    printf("2000     김민수   컴퓨터학부    3.0₩n");
    printf("2011     홍길동   전자공학부    4.3₩n");
    print_line();

    return 0;
}
```

함수

3. 매개변수와 반환 값이 모두 없는 함수

```
15
16  void print_line() {
17      for (int i = 0; i < 37; i++) {
18          printf("-");
19      }
20
21      printf("₩n");
22  }
```

함수

4. 재귀호출 함수

재귀 호출(recursive call)이란?

함수 내부에서 함수가 자기 자신을 또 다시 호출하는 행위.
자기가 자신을 계속 호출하기 때문에 함수 내에서 재귀 호출을 중단하도록 조건 명령문을 반드시 작성해야한다.

탈출 조건이 없는 경우 프로그램이 사용할 수 있는 메모리를 모두 사용할 때까지 무한 반복된다.

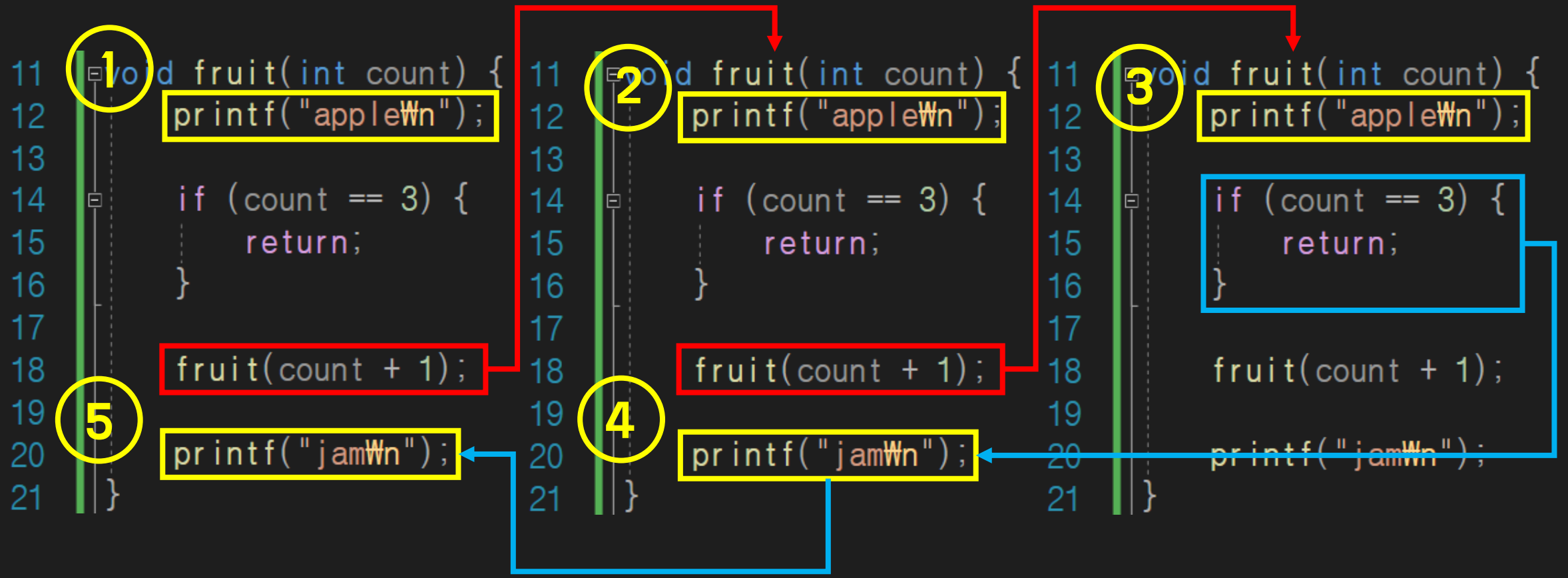
함수

4. 재귀호출 함수

```c
#include <stdio.h>

void fruit(int count);

int main() {
    fruit(1);

    return 0;
}
```

```c
void fruit(int count) {
    printf("apple₩n");

    if (count == 3) {
        return;
    }

    fruit(count + 1);

    printf("jam₩n");
}
```

실행 결과는??


1
2
3
4
5
11 void fruit(int count) {
12 printf("apple₩n");
13
14 if (count == 3) {
15 return;
16 }
17
18 fruit(count + 1);
19
20 printf("jam₩n");
21 }

문제 해결

백준 단계별로 풀어보기 6단계 (https://www.acmicpc.net/step)

"심화" 2번(3003), 3번(2444), 7번(4344)

단계	제목	설명
1	입출력과 사칙연산	입력, 출력과 사칙연산을 연습해 봅시다. Hello World!
2	조건문	if 등의 조건문을 사용해 봅시다.
3	반복문	for, while 등의 반복문을 사용해 봅시다.
4	1차원 배열	배열을 사용해 봅시다.
5	문자열	문자열을 다루는 문제들을 해결해 봅시다.
6	심화 1	지금까지의 프로그래밍 문법으로 더 어려운 문제들을 풀어봅시다.
7	2차원 배열	배열 안에 배열이 있다면 어떨까요? 2차원 배열을 만들어 봅시다.

해결 못한 문제는 다음 멘토링 시간 전까지 해오기

끝