C언어 기초 part. 2

Week 1 – 함수, 배열 기본

QnA 메일 : edujongkook@gmail.com

Pdf 파일 : github.com/edujongkook
/pdf_sbs_c_weekend

목차 A table of contents

복습 if / else if / else 문

```
if (조건식 1) {
  조건식1이 참인 경우 실행되는 코드
}
else if (조건식 2) {
 조건식1이 거짓 조건식 2가 참인경우 실행되는 코드
}
else {
 모든 조건식이 거짓인 경우 실행되는 코드
}
```

복습 if / else if / else 문

```
int main(void) {
    int speed; // 차량의 속도
    scanf_s("%d", &speed);
    if (speed > 100) {
        printf("과속입니다.₩n");
    }
    else if(speed < 50){
        printf("너무 속도가 느립니다.₩n");
    }
    else {
        printf("정상속도 입니다.₩n");
    }
}
```

복습 switch문의 구조

```
switch (비교값) {
case 값1 :
        (비교값 == 값1) 인 경우 실행되는 코드
        break;
default:
        비교값이 어떤 case에도 맞지 않는 경우 실행되는 코드
}
```

복습 switch문의 구조

```
int main(void) {
    int n = 10;
    switch (n) {
    case 10:
        printf("10₩n");
        break;
    case 100:
        printf("100₩n");
        break;
    default:
        printf("default₩n");
    }
```

복습 while문의 구조

```
while (조건식) {
        조건식이 참인경우 반복되는 코드
}
```

```
do {
        무조건 한번 실행 후 조건식이 참인경우 반복되는 코드
} while (조건식) ;
```

복습 숫자 맞추기 do while반복

```c
int target = 70; // 정답 숫자
int num;
do {
    printf("숫자를 맞춰보세요 : ");
    scanf_s("%d", &num);
    if (target > num) {
        puts("UP");
    }
    else if (target < num) {
        puts("DOWN");
    }
    else if (target == num) {
        puts("숫자를 맞췄습니다!!!");
    }
} while (target != num);
```

무조건 한번 do 블록이

실행 된 이후에

조건문을 검사하여 반복

복습 for 문의 구조

```
for (초기값 ; 조건식 ; 증감식)
{
        조건식이 참일 때 반복할 코드
}
```

복습 구구단 출력

2 x 1 부터 2 x 9 까지 for 문을 이용하여 출력하는 코드입니다.

```
int main(void) {
    int dan; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    dan = 2;
    for (mul = 1; mul < 10; mul++){
        printf("%d x %d = %d₩n",dan, mul, dan * mul);
    }
}
```

복습 반복문의 break

반복문에서 break 를 사용하면 반복문을 탈출합니다.

```
int main(void) {
    int dan; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    dan = 2;
    for (mul = 1; mul < 10; mul++){
        if (mul == 7) {
            break;
        }
        printf("%d x %d = %d₩n",dan, mul, dan * mul);
    }
}
```

복습 반복문의 continue

반복문에서 continue를 사용하면 반복문의 처음으로 돌아갑니다.

```
int main(void) {
    int dan; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    dan = 2;
    for (mul = 1; mul < 10; mul++){
        if (mul == 5) {
            continue;
        }
        printf("%d x %d = %d₩n",dan, mul, dan * mul);
    }
}
```

복습 for문의 중첩

블록 { … } 을 생성하는 모든 문법들은 중첩될 수 있습니다.

이를 이용하여 2x1 부터 9x9 까지 쉽게 출력할 수 있습니다.

```c
int main(void) {
    int dan; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    for (dan = 2; dan < 10; dan++) {
        for (mul = 1; mul < 10; mul++) {
            printf("%d x %d = %d₩n", dan, mul, dan * mul);
        }
    }
}
```

복습 for문의 중첩

중첩에서 break / continue의 범위 – 가장 가까운 반복문에만 영향을 미칩니다.

```
int main(void) {
    int dan; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    for (dan = 2; dan < 10; dan++) {
        for (mul = 1; mul < 10; mul++) {
            if(mul == 5){
                break;
            }
            printf("%d x %d = %d₩n", dan, mul, dan * mul);
        }
    }
}
```

1.

함수


IVE THE LIFE YOU'VE
LWAYS DREAMED OF.
BE FEARLESS IN THE
FACE OF ADVERSITY.
EVER STOP LEARNING.
SE YOUR IMAGINATION
WHENEVER POSSIBLE.
ECOGNIZE THE BE AUTY
THAT SURROUN
REMEMBER WHI
AME FROM, BU
LOSE SIGHT OF
YOU ARE GO

1 함수의 구조

```
리턴타입 함수이름 (매개변수타입 매개변수이름) {
...
return 리턴값;
}
```

```
int main(void) {
    int dan;
    dan = 2;
    return 0;
}
```

1 기본적인 함수 구조

이전에 main에 있던 구구단 코드를 함수로 만들어 호출해 보겠습니다.

```
int main(void) {
    gugu();
}

int gugu(void) {
    int dan = 2; // 구구단의 단
    int mul; // 단에 곱해질 수 (1 ~ 9)
    for (mul = 1; mul < 10; mul++) {
        printf("%d x %d = %d₩n", dan, mul, dan * mul);
    }
    return 0;
}
```

1 함수 선언과 호출

```
int main(void) {
    int gugu(void);
    gugu();
}
```

변수와 마찬가지로

사용하기 전에 먼저 선언

1 return 값의 사용

함수를 호출 하면

함수가 실행되고 리턴값이 들어옵니다.

```
int main(void) {
    int gugu(void), ret;
    ret = gugu();
    printf("ret : %d₩n", ret);
}
```

```
int gugu(void) {
    int dan = 2; // 구
    int mul; // 단에 곱
    for (mul = 1; mul
        printf("%d x %
    }
    return 0;
}
```

1 return 값의 사용

함수도 변수처럼 다양한 곳에서 사용 가능

```
int main(void) {
    int gugu(void);
    printf("ret : %d₩n", gugu());
}
```

```
int gugu(void) {
    int dan = 2; // 구
    int mul; // 단에 곱
    for (mul = 1; mul
        printf("%d x %
    }
    return 0;
}
```

1 return 값의 사용

어떤 값을 리턴하는지에 따라 값은 변경됨

```c
int main(void) {
    int gugu(void);
    printf("ret : %d₩n", gugu());
}
```

```c
int gugu(void) {
    int dan = 2; // 구구단
    int mul; // 단에 곱해
    for (mul = 1; mul < 1
        printf("%d x %d =
    }
    return 50;
}
```

1 함수의 선언 위치

보통은 변수와 선언을 함께하면 구분이 어려워

선언은 메인함수 위에서 먼저 이루어집니다.

```
int gugu(void);

int main(void) {
    int ret;
    ret = gugu();
    printf("ret : %d₩n", ret);
}
```

1 함수의 매개변수

함수에 전달된 값이 저장되는 변수를 매개변수라고 합니다.

```
void gugu(int dan) {
    int mul; // 단에 곱해질 수 (1 ~ 9)
    for (mul = 1; mul < 10; mul++) {
        printf("%d x %d = %d₩n", dan, mul, dan * mul);
    }
}
```

1 함수의 매개변수

선언과 호출도 함수 형태와 동일하게 바꿔줍니다.

```
void gugu(int dan);

int main(void) {
    int num = 2;
    gugu(num);
}
```

1 함수의 매개변수

호출하는 곳에서 함수에게 매개변수를 통해 값이 전달됩니다.

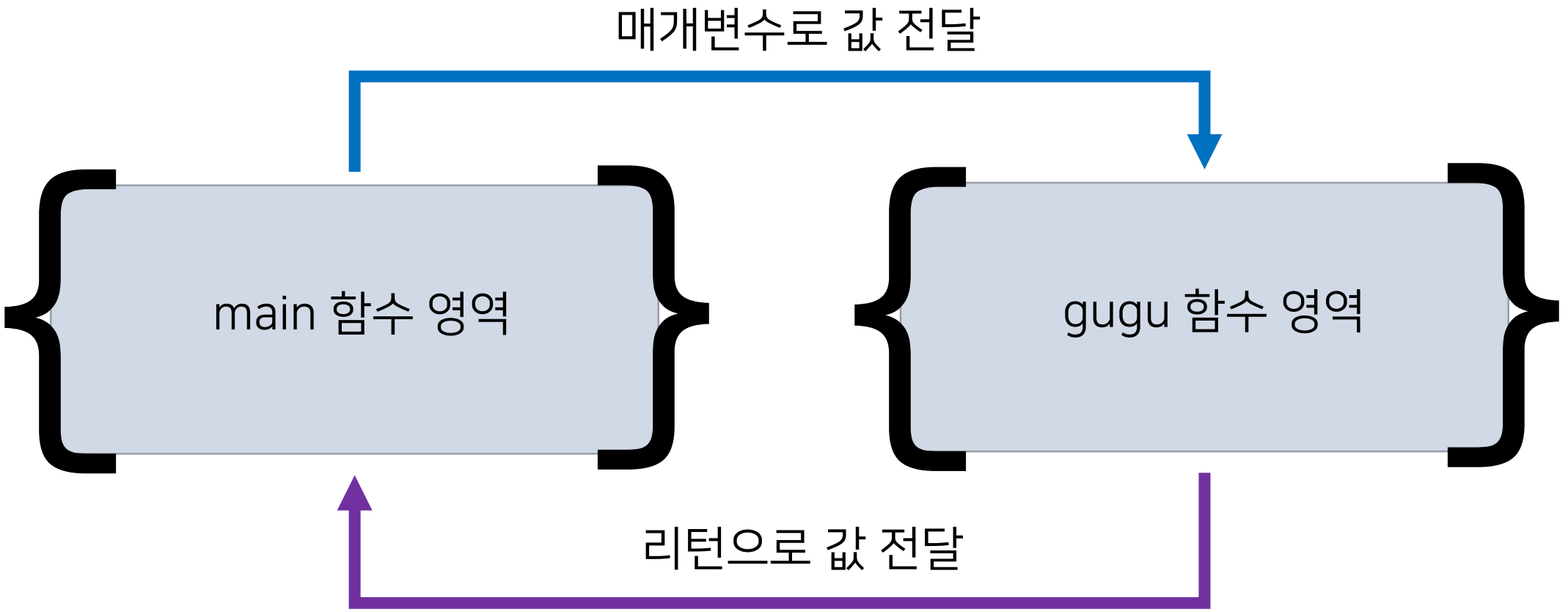
```
void gugu(int dan);

int main(void) {
    int num = 2;
    gugu(num);
}
```

```
void gugu(int dan) {
    int mul; // 단에 곱해질 수 (1 ~
    for (mul = 1; mul < 10; mul++) {
        printf("%d x %d = %d₩n", dan
    }
}
```

1 함수의 매개변수

매개변수와 리턴으로 값을 주고 받을 수 있습니다.



1 함수의 매개변수

매개변수는 여러 개를 전달할 수 있습니다.

전달된 두 수의 합을 계산하는 함수를 새로 만들어 보겠습니다.

```
int sum(int num1, int num2) {
    int result = num1 + num2;
    return result;
}
```

1 함수의 매개변수

함수의 선언과 호출 부분 / 두 수를 입력 받아 함수에 전달합니다.

```
int sum(int num1, int num2);

int main(void) {
    int num1, num2;
    puts("더해줄 두 수를 입력해 주세요");
    scanf_s("%d", &num1);
    scanf_s("%d", &num2);
    printf("계산결과 : %d₩n", sum(num1, num2));
}
```

1 사칙연산 계산 함수 (선언과 호출)

두 수에 추가로 연산자 기호를 더하여 이전에 만들어봤던
간단한 계산기 함수를 만들어 보겠습니다.

```
int calc(int num1, int num2, char opt);

int main(void) {
    int num1, num2;
    char opt;
    puts("계산식을 입력해 주세요 예)4 + 2");
    scanf_s("%d %c %d", &num1, &opt, 1, &num2);
    printf("계산결과 : %d₩n", calc(num1, num2, opt));
}
```

1 사칙연산 계산 함수 (정의 부분)

```
int calc(int num1, int num2, char opt) {
    int result;
    switch (opt) {
    case '+':
        result = num1 + num2;
        return result;
    case '-':
        result = num1 - num2;
        return result;
        // 곱하기, 나누기 생략
    }
}
```

1 사칙연산 계산 함수 (정의 부분)

```
int calc(int num1, int num2, char opt) {
    int result;
    switch (opt) {
    case '+':
        result = num1 + num2;
        return result;
    case '-':
        result = num1 - num2;
        return result;
        // 곱하기, 나누기 생략
    }
}
```

1 scanf의 엔터키 입력문제

수식을 이런식으로 입력받는 이유는 scanf_s로 문자를 입력받을때 엔터키를 문자로 취급하기 때문입니다.

```
int main(void) {
    int num1, num2;
    char opt;
    puts("숫자1을 입력하세요 :");
    scanf_s("%d", &num1);
    puts("연산자를 입력하세요 :");
    scanf_s("%c", &opt, 1);
    puts("숫자2을 입력하세요 :");
    scanf_s("%d", &num2);
    printf("계산결과 : %d₩n", calc(num1, num2, opt));
}
```

이 부분에서 숫자1를 입력하고 엔터키를 칠때 엔터키가 문자로 입력되어 버립니다.

1 scanf의 엔터키 입력문제

scanf의 이런 동작방식 때문에 원하는 결과를 위해 다음과 같은 처리를 해주거나 다른 입력함수(getch , kbhit) 등 을 사용할 수 있음.

```
int main(void) {
    int num1, num2;
    char opt;
    puts("숫자1을 입력하세요 :");
    scanf_s("%d", &num1);
    rewind(stdin);
    puts("연산자를 입력하세요 :");
    scanf_s("%c", &opt, 1);
    puts("숫자2을 입력하세요 :");
    scanf_s("%d", &num2);
    printf("계산결과 : %d₩n", calc(num1, num2, opt));
}
```

입력된 엔터키 문자를 아래 scanf_s에서 입력받기 전에 초기화 시켜준다.

rewind 함수는 키보드로 입력된 (stdin) 임시저장공간을 초기화

2.

변수의 범위

2 블록영역 내부의 변수

이제는 main 함수의 블록말고 다른 함수의 블록이 등장했습니다.

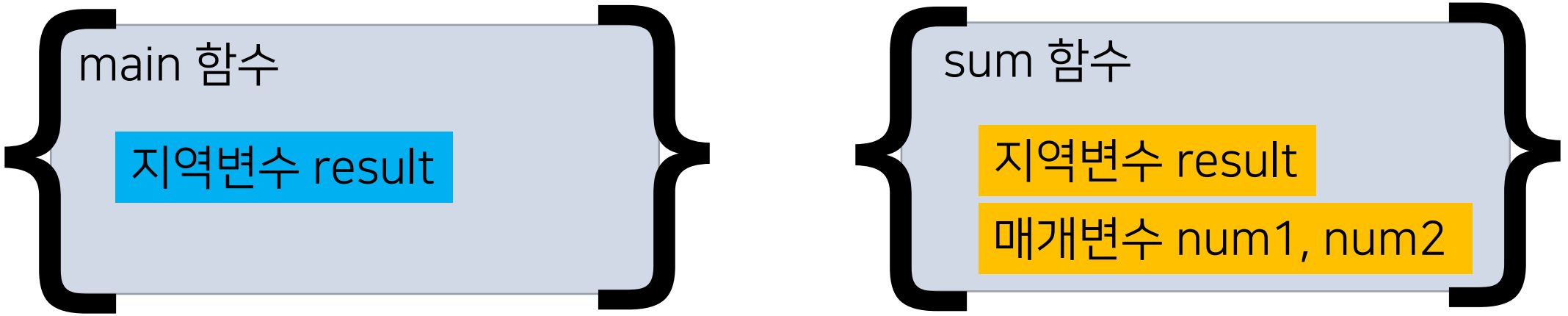
각각의 블록에서 선언한 변수는 그 블록 안에서만 사용 가능합니다.

```
int main(void) {
    int result = 100;
    printf("main함수의 result : %d₩n", result);
    sum(10, 20);
}
int sum(int num1, int num2) {
    int result = num1 + num2;
    printf("sum함수의 result : %d₩n", result);
    return result;
}
```

블록영역 내부의 변수

이렇게 각 블록 영역내에서만 사용되는 변수를

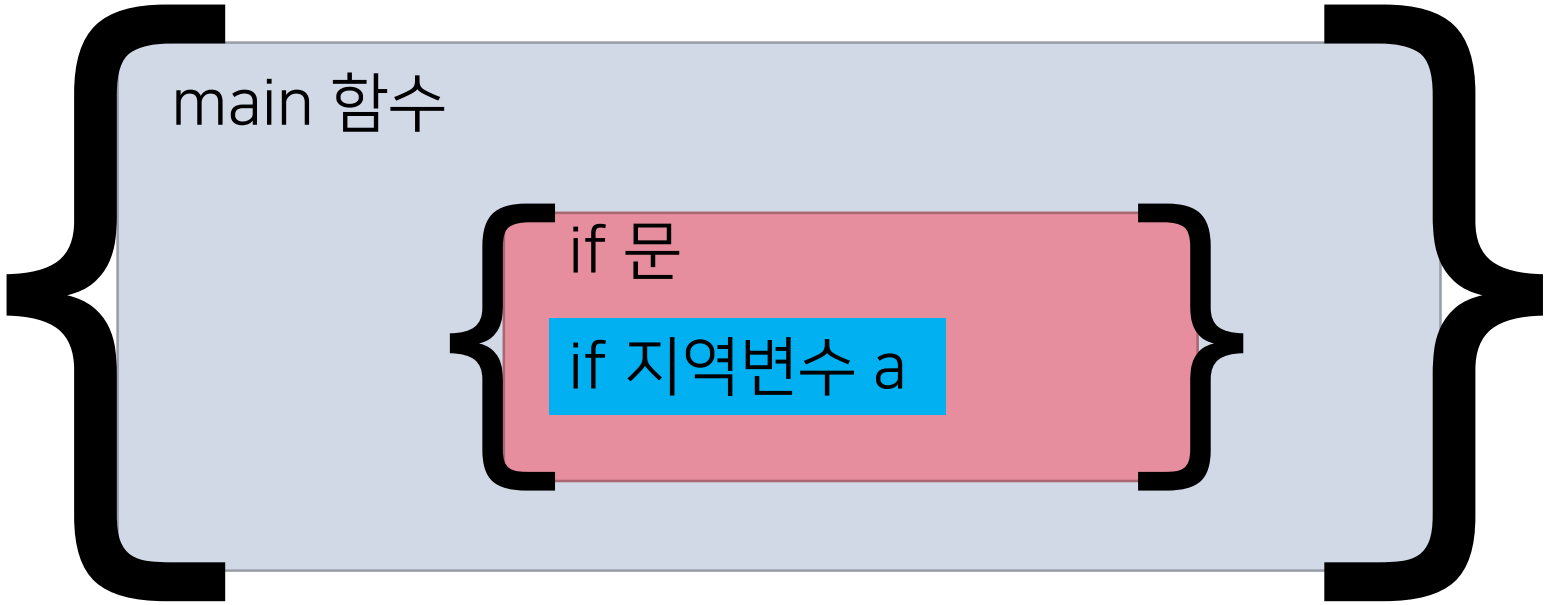
지역변수 (local variable) 라고 부릅니다. (매개변수도 특수한 지역변수 입니다)



2 블록영역 내부의 변수

만약에 함수 안에서 블록을 만들고 (예를들어 if 블록) 지역변수를 선언하면
그 지역변수는 해당 if 블록 밖에서는 사용할 수 없습니다. (안쪽에서만 사용가능)

```c
int main(void) {
    if (1) {
        int a = 0;
    }
    printf("%d", a);
}
```



2 블록영역 밖의 변수

반면에 아래처럼 어느 블록영역에도 해당하지 않는 변수를 선언할 수 있습니다.

```
void gugu(int dan);
int sum(int num1, int num2);
int calc(int num1, int num2, char opt);
int result = 100;

int main(void) {
    printf("main함수에서 result : %d₩n", result);
```

2 블록영역 밖의 변수

서로 다른 블록에서 result 를 출력해 보겠습니다.

이렇게 블록 밖에서 만들어진 변수를 **전역변수**(global variable) 라고 합니다.

```
int result = 100;

int main(void) {
    printf("main함수에서 result : %d₩n", result);
    sum(10, 20);
}
int sum(int num1, int num2) {
    printf("sum함수에서 result : %d₩n", result);
    return result;
}
```

2 블록영역 밖의 변수

전역변수를 사용하면 return 값 없이도 동일한 결과를 만들 수 있습니다.

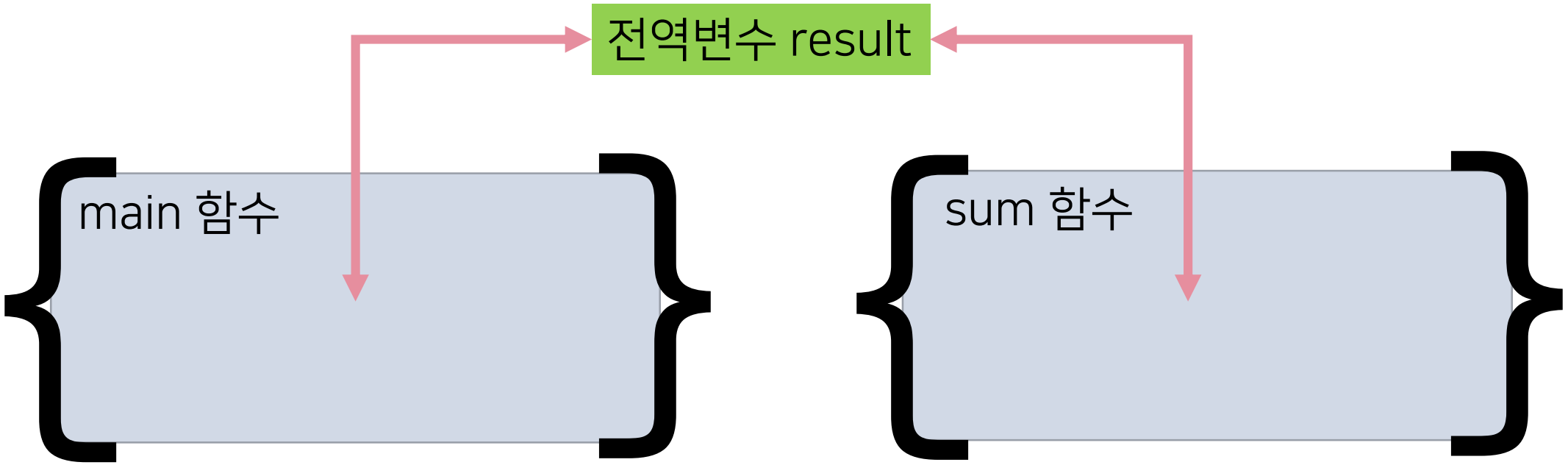
```
int result;

int main(void) {
    sum(10, 20);
    printf("계산결과 : %d₩n", result);
}

int sum(int num1, int num2) {
    result = num1 + num2;
}
```

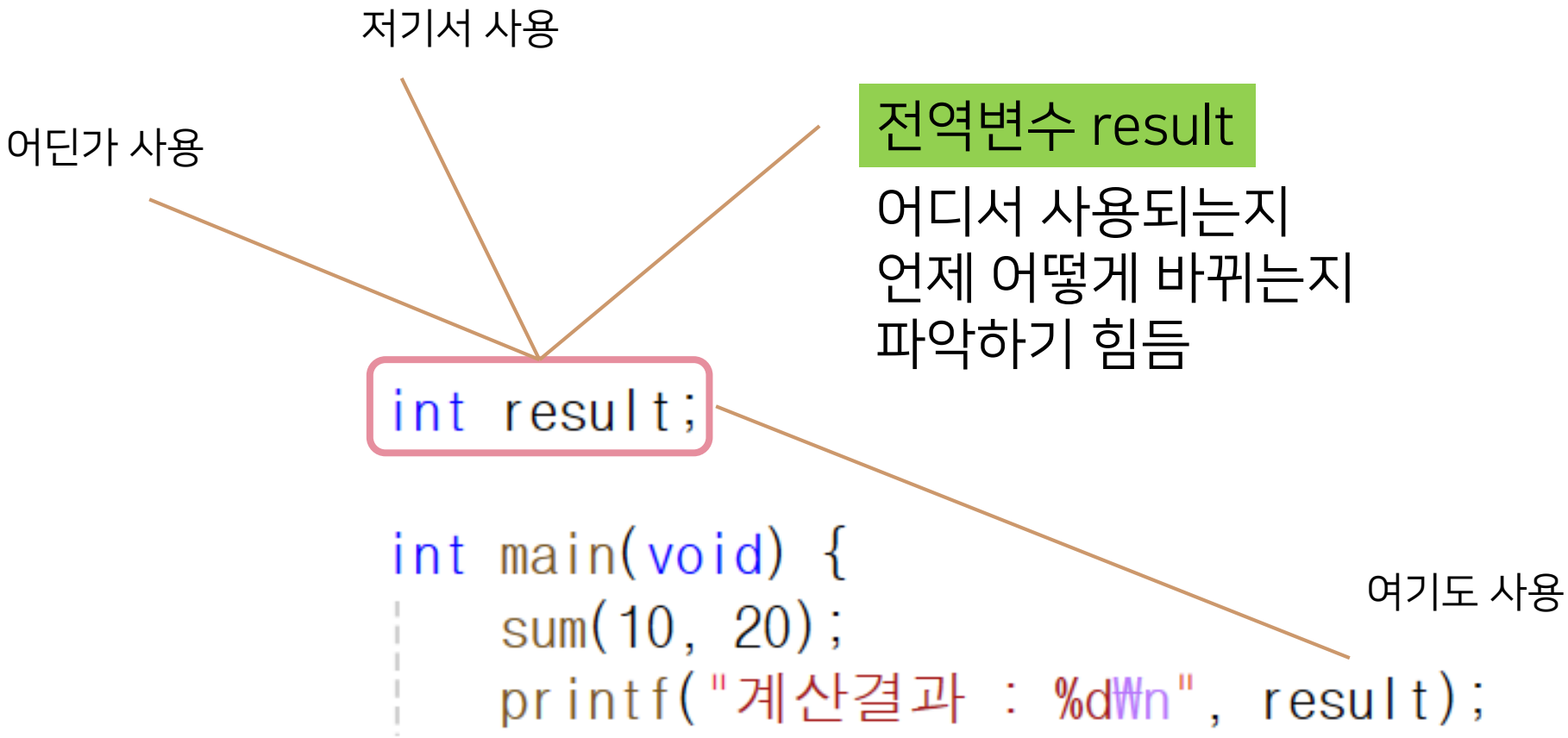
블록영역 내부의 변수

전역변수는 해당 소스파일의 어느 곳에서나 사용 가능



2 블록영역 밖의 변수

하지만 코드가 복잡해 질수록 전역변수는 관리하기 어려워 집니다.



2 블록영역 밖의 변수

따라서 전역변수는 특별한 경우에만 사용하고

보통의 경우는 지역변수와 매개변수, 리턴값을 사용해 코딩하는 것이 좋습니다.

```
int main(void) {
    printf("계산결과 : %d₩n", sum(10, 20));
}
int sum(int num1, int num2) {
    return num1 + num2;
}
```

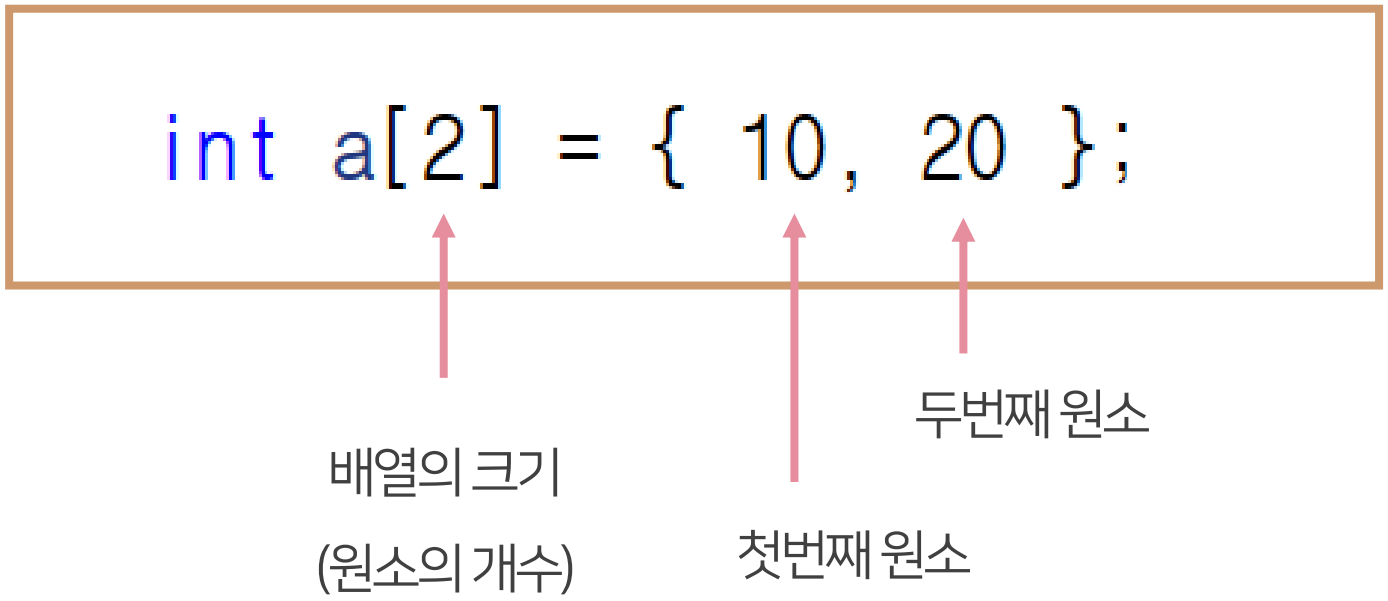
3.

배열과 문자열

3 배열의 구조

순서가 있는 값들을 배열(array)이라고 합니다.

int 타입의 2개의 원소를 가지고있는 배열 a를 만드는 법은 아래와 같습니다.



3 배열의 선언과 초기화

변수와 마찬가지로 선언을 하고 나중에 값을 넣을 수도 있습니다.

선언만 할 때는 반드시 크기를 정해 선언해야 합니다.

```
int main(void) {
    int a[100]; // 먼저 선언만 하고 나중에 초기화 가능
    int b[]; // ERROR 선언만 하는경우 크기지정 필수
    int c[] = { 100, 200, 300 }; // 초기화 하는경우 크기생략가능
}
```

3 배열의 사용

배열의 원소를 사용하기 위해서 인덱스를 사용합니다.

인덱스는 0부터 시작하여 1씩 증가합니다.

```
int main(void) {
    int a[2] = { 10, 20 };
    printf("a의 첫번째 원소 : %d₩n", a[0]);
    a[0] = 100; // 첫번째 원소를 100으로 변경
    printf("a[0] 변경 후 : %d₩n", a[0]);
}
```

인덱스(index)

3 배열과 반복문

배열의 크기가 작으면 직접 만들 수 있겠지만

```
int main(void) {
    // 1부터 5까지 숫자로 이루어진 배열
    int number[5] = { 1, 2, 3, 4, 5 };
    int number2[100]; // 100까지 숫자를 넣으려면..
}
```

3 배열과 반복문

원소가 많아지면 반복문을 사용해야 합니다.

```
int main(void) {
    int number[100]; // 1부터 100까지 숫자로 이루어진 배열
    for (int i = 0; i < 100; i++) {
        number[i] = i + 1;
    }
}
```

3 문자열

문자열은 문자형 값들로 이루어진 배열입니다.

```
int main(void) {
    char c = 'a';
    char str[6] = { 'a', 'p', 'p', 'l', 'e', '₩0' };
    printf("문자형 변수 c : %c₩n", c);
    printf("문자형 배열 (문자열) str : %s₩n", str);
}
```

3 문자열

다양한 방식으로 선언하고 초기화 할 수 있습니다.

마찬가지로 선언만 할 때는 크기를 지정해 주어야 합니다.

```
int main(void) {
    char str[6] = { 'a', 'p', 'p', 'l', 'e', '₩0' };
    char str2[] = "apple";
    char str3[6];  // 먼저 선언만 값 할당은 나중에
    printf("str2 의 크기 : %d₩n", sizeof(str2));
}
```

3 문자열

문자열의 끝을 의미하는 '₩0'라는 특수문자 (null 문자) 가 들어갑니다.

str2 의 방식으로 값을 넣을 때도 실제로는 마지막에 널문자가 자동포함.

```
int main(void) {
    char str[6] = { 'a', 'p', 'p', 'l', 'e', '₩0' };
    char str2[] = "apple";
    char str3[6];  // 먼저 선언만 값 할당은 나중에
    printf("str2 의 크기 : %d₩n", sizeof(str2));
}
```

3 배열을 함수의 매개변수로 사용

변수처럼 배열을 함수로 전달할 수 있습니다.

배열을 매개변수로 받는 함수 sum_array 의 선언과 호출부분

```
int sum_array(int nums[]);

int main(void) {
    int nums[] = { 10, 20, 30 };
    int ret = sum_array(nums);
    printf("sum_array 리턴 값 : %d₩n", ret);
}
```

3 배열을 함수의 매개변수로 사용

배열을 매개변수로 받는 함수 sum_array 의 정의 부분

```
int sum_array(int nums[]) {
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += nums[i];
    }
    return sum;
}
```

3 배열을 함수의 매개변수로 사용

매개변수로 배열 그 자체를 전달하는 것이 아닙니다.

배열의 포인터를 전달합니다. (포인터 시간에 자세히 다루겠습니다.)

```
int sum_array(int nums[]) {
    int sum = 0;
    for (int i = 0; i < 3; i++) {
        sum += nums[i];
    }
    printf("nums는 사실 배열의 주소(포인터) 입니다. %p₩n", nums);
    return sum;
}
```