12강

함수와 포인터 활용

동양미래대학교 **강환수** 교수

본 강의 사용 및 참조 자료

- Perfect C, 3판, 강환수 외 2인 공저, 인피니티북스, 2021

14장 함수와 포인터 활용

목차

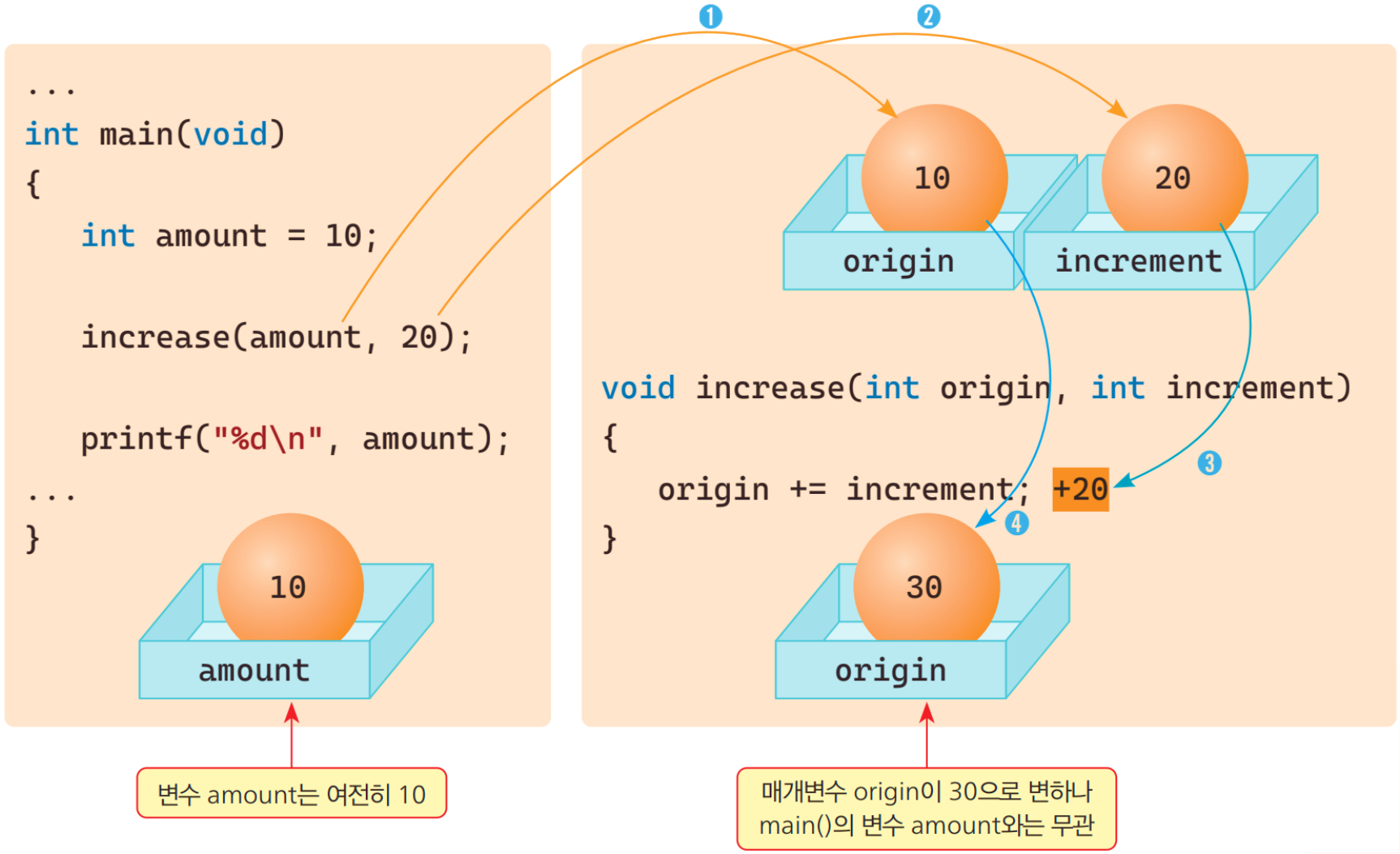
01

함수의 인자전달 방식

함수에서 값의 전달

- 값에 의한 호출(call by value) 방식
  - 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미
- 함수 increase(int origin, int increment) 함수 호출 시
  - origin += increment; 를 수행하는 함수
    - 변수 amount의 값 10이 매개변수 origin에 복사
    - 상수 20이 매개변수 increment에 복사

값에 의한 호출 1/2



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 598쪽

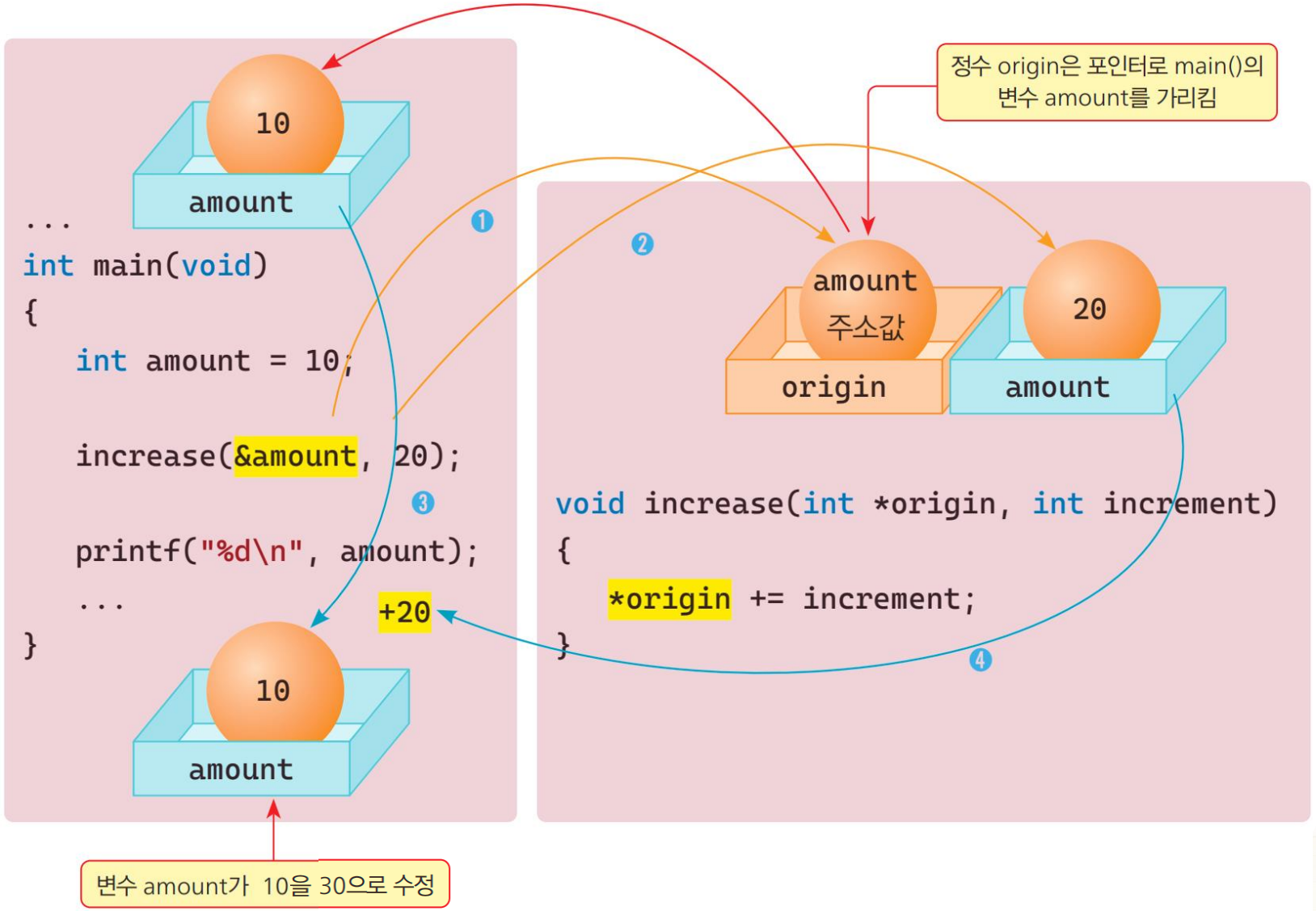
값에 의한 호출 2/2

- 함수 increase(int origin, int increment) 함수 호출 시
  - 함수 increase( ) 내부 실행
    - 매개변수인 origin 값이 30으로 증가
  - 변수 amount와 매개변수 origin은 아무 관련성이 없음
    - origin은 증가해도 amount의 값은 변하지 않음
  - 값에 의한 호출(call by value) 방식
    - 함수 외부의 변수를 함수 내부에서 수정할 수 없는 특징

주소에 의한 호출(call by address) 1/3

- 포인터를 매개변수로 사용
  - 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 가능

주소에 의한 호출(call by address) 2/3



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 599쪽

주소에 의한 호출(call by address) 3/3

- increase(int *origin, int increment)
  - 첫 번째 매개변수를 int *
    - 함수 구현도 *origin += increment;로 수정하여 구현
  - 함수 호출 시 첫 번째 인자가 &amount
    - 변수 amount의 주소 값이 매개변수인 origin에 복사
    - 상수 20이 매개변수인 increment에 복사
  - 함수 increase( ) 내부 실행
    - *origin은 변수 amount 자체를 의미
    - *origin을 증가시키면 amount의 값도 증가
  - main( ) 내부에서 amount의 값이 30으로 증가

배열이름으로 전달

- 배열을 매개변수로 하는 함수 sum( )을 구현
  - 실수형 배열의 모든 원소의 합을 구하여 반환하는 함수
    - 함수 sum( )의 형식매개변수는 실수형 배열과 배열크기
  - 매개변수에서 배열크기를 기술하는 것은 아무 의미가 없음
    - double ary [5] 보다는 double ary [ ] 라고 기술하는 것을 권장
      - 실제로 함수 내부에서 실인자로 전달된 배열의 배열크기를 알 수 없음
    - 배열크기를 두 번째 인자로 사용
  - 매개변수를 double ary [ ] 처럼 기술해도
    - 단순히 double *ary처럼 포인터 변수로 인식

함수에서 배열 전달을 위한 함수원형

함수원형과 함수 호출

```
double sum(double ary[], int n);
//double sum(double [], n); 가능

    ...

double data[] = {2.3, 3.4, 4.5, 6.7, 9.2};

    ... sum(data, 5);
```

함수 호출 시 배열이름으로 배열인자를 명시한다.

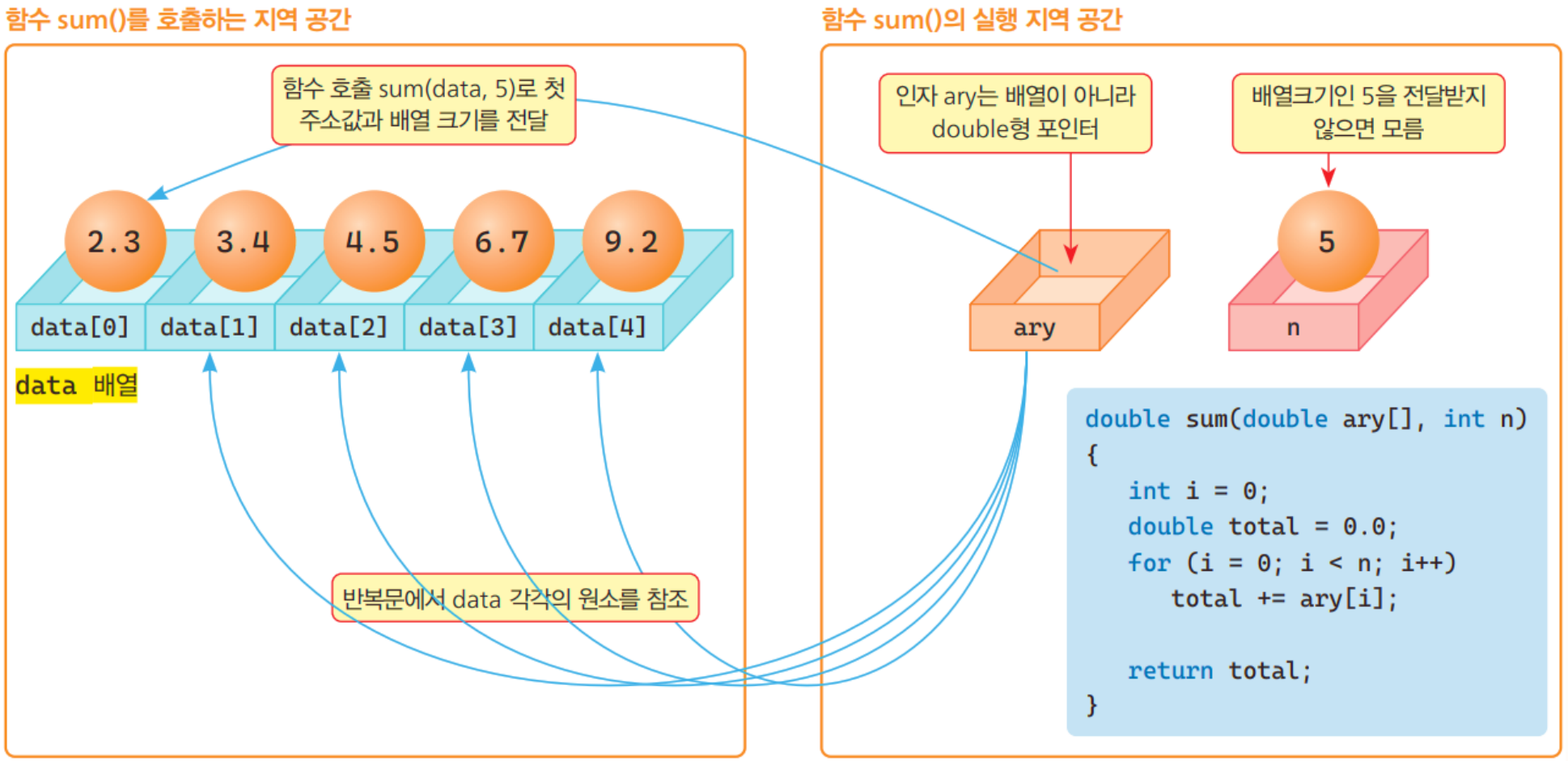
함수 정의

```
double sum(double ary[], int n)
{
    int i = 0;
    double total = 0.0;
    for (i = 0; i < n; i++)
        total += ary[i];

    return total;
}
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 601쪽

배열 전달을 위한 함수호출



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 602쪽

다양한 배열원소 참조 방법

- 1차원 배열 point에서
  - 간접연산자를 사용한 배열원소의 접근 방법은 *(point + i)
    - 배열의 합을 구하려면 sum += *(point + i); 문장을 반복
  - 문장 int *address = point;
    - int 포인터 포인터 변수 address를 선언하여 배열 point를 저장
    - 문장 sum += *(address++)으로도 배열의 합 가능
  - 배열이름 point는 주소 상수
    - sum += *(point++)는 사용 불가능
      - 배열 이름인 point는 상수로 변화되는 주소 값을 저장할 수 없기 때문

간접연산자 *를 사용한 배열원소의 참조방법

```
int i, sum = 0;
int point[] = {95, 88, 76, 54, 85, 33, 65, 78, 99, 82};
int *address = point;
int aryLength = sizeof (point) / sizeof (int);
```

가능

```
for (i=0; i<aryLength; i++)
  sum += *(point+i);
```

가능

```
for (i=0; i<aryLength; i++)
   sum += *(address++);
```

오류

```
for (i=0; i<aryLength; i++)
  sum += *(point++);
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 602쪽

형식 매개변수 int ary[]와 int *ary

- 함수 머리에 배열을 인자로 기술하는 다양한 방법
  - 함수헤더에 int ary[ ]로 기술하는 것은 int *ary로도 대체 가능
- for 문의 반복몸체 블록
  - 배열 원소의 합을 구하는 문장 4개
  - 변수 ary는 포인터 변수로서 주소 값을 저장하는 변수
    - 증가연산자의 이용 가능
    - 연산식 *ary++
      - *(ary++)와 같은 의미

함수에서 int 주소 값 저장 매개변수

같은 의미로 모두 사용할 수 있다

```
int sumary(int ary[], int SIZE)
{
    ...
}
```

```
int sumaryf(int *ary, int SIZE)
{
    ...
}
```

```
for (i = 0; i < SIZE; i++)
{
    sum += ary[i];
}
```

```
for (i = 0; i < SIZE; i++)
{
    sum += *(ary + i);
}
```

```
for (i = 0; i < SIZE; i++)
{
    sum += *ary++;
}
```

```
for (i = 0; i < SIZE; i++)
{
    sum += *(ary++);
}
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 603쪽

02

포인터 전달과 반환

매개변수와 반환으로 포인터 사용 1/2

- 주소연산자 &
  - 함수에서 매개변수를 포인터로 이용하면 결국 주소에 의한 호출
  - 함수원형 void add(int *, int, int);
    - 첫 매개변수가 포인터인 int *
    - 내부 구현
      - 두 번째와 세 번째 인자를 합해 첫 번째 인자가 가리키는 변수에 저장
      - 변수인 sum을 선언하여 주소값인 &sum을 인자로 호출

매개변수와 반환으로 포인터 사용 2/2

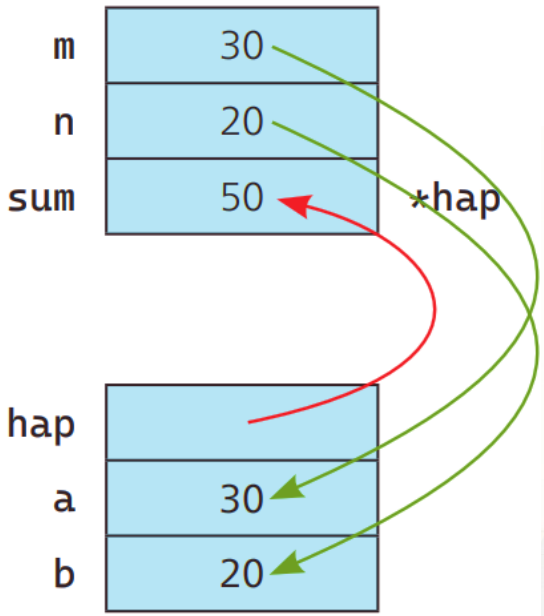
```c
int m = 0, n = 0, sum = 0;
scanf("%d %d", &m, &n);

add(&sum, m, n);
```

함수 호출 후 변수 sum에는 m과 n의 합이 저장

```c
void add(int *hap, int a, int b)
{
    *hap = a + b;
}
```

함수 add() 정의에서 합을 저장하는 문장



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 611쪽

실습예제 1/2

Prj04 04ptrparam.c 함수로 포인터를 전달하는 주소에 의한 호출 난이도: ★★

```
01  #define _CRT_SECURE_NO_WARNINGS
02  #include <stdio.h>
03
04  void add(int*, int, int);
05
06  int main(void)
07  {
08      int m = 0, n = 0, sum = 0;
09
10      printf("두 정수 입력: ");
11      scanf("%d %d", &m, &n);
12      add(&sum, m, n);
```

m과 n을 더한 결과가 변수 sum에 저장

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 611쪽

실습예제 2/2

```
13      printf("두 정수 합: %d\n", sum);
14
15      return 0;
16  }
17
18  void add(int* psum, int a, int b)
19  {
20      *psum = a + b;
21  }
```

```
두 정수 입력: 10 20
두 정수 합: 30
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 612쪽

함수의 결과를 포인터로 반환 1/2

- 함수 add( )의 함수원형
  - int * add (int *, int, int)
    - 반환값이 포인터인 int *
    - 두 수의 합을 첫 번째 인자가 가리키는 변수에 저장한 후
    - 포인터인 첫 번째 인자를 그대로 반환
- add( )를 *add(&sum, m, n) 호출
  - 변수 sum에 합 a+b가 저장
  - 반환값인 포인터가 가리키는 변수인 sum을 바로 참조

함수의 결과를 포인터로 반환 2/2

```
int * add(int *, int, int);

    int m = 0, n = 0, sum = 0;
    ...
    scanf("%d %d", &m, &n);

    printf("두 정수 합: %d\n", *add(&sum, m, n));
```

```
int * add(int *psum, int a, int b)
{
    *psum = a + b;
    return psum;
}
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 612쪽

상수를 위한 const 사용

- 포인터를 매개변수로 이용하면 수정된 결과를 받을 수 있어 편리
  - 이러한 포인터 인자의 잘못된 수정을 미리 예방하는 방법
    - 수정을 원하지 않는 함수의 인자 앞에 키워드 const를 삽입
    - 참조되는 변수가 수정될 수 없게 함
  - 키워드 const는 인자인 포인터 변수
    - 가리키는 내용을 수정 불가능

const double *a, const double *b

- *a와 *b를 대입연산자의 l-value로 사용 불가능
  - 즉 *a와 *b를 이용하여 그 내용을 수정 불가능
- 상수 키워드 const의 위치
  - 자료형 앞이나 포인터변수 *a 앞에도 가능
  - const double *a와 double const *a 는 동일한 표현

const 인자의 이용

```
// 매개변수 포인터 a, b가 가리키는 변수의 내용은 수정하지 못함
void multiply(double *result, const double *a, const double *b)
{
    *result = *a * *b;
    //다음 두 문장은 오류 발생
    *a = *a + 1;
    *b = *b + 1;
}
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 615쪽

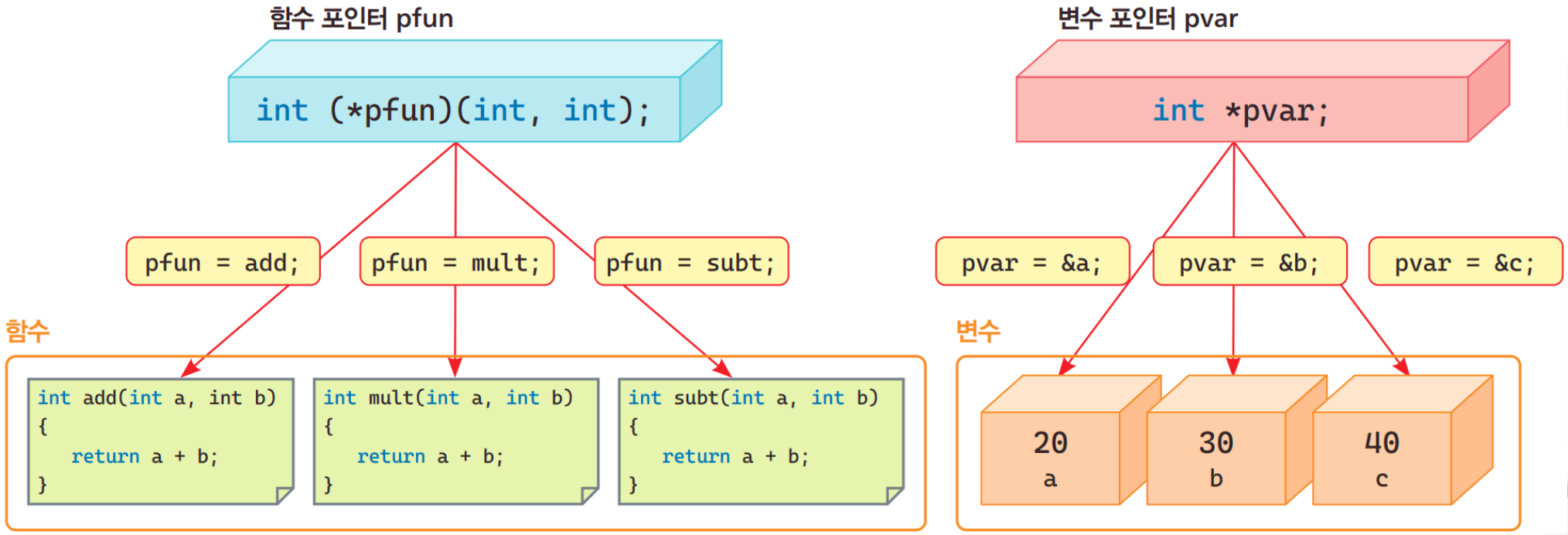
03

함수 포인터와 void 포인터

함수 포인터 개요

- 함수 주소 저장 변수
  - 포인터의 장점은 다른 변수를 참조하여 읽거나 쓰는 것도 가능
- 함수 포인터
  - 하나의 함수 이름으로 필요에 따라 여러 함수를 사용하면 편리
  - 함수 포인터 pfun
    - 함수 add( )와 mult( ) 그리고 subt( )로도 사용 가능

함수 포인터 활용



[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 623쪽

함수 포인터 변수 선언 1/2

- 함수 포인터(pointer to function)
  - 함수의 주소값을 저장하는 포인터 변수
    - 즉 함수를 가리키는 포인터
  - 함수의 주소를 저장할 수 있는 변수
    - 반환형, 인자목록의 수와 각각의 자료형이 일치
  - 함수 포인터 선언
    - 함수원형에서 함수이름을 제외한 반환형과 인자목록의 정보가 필요

함수 포인터 변수 선언 1/2

함수 포인터 변수 선언

```
반환자료형  (*함수 포인터변수이름)(자료형1 매개변수이름1, 자료형2 매개변수이름2, ...);
반환자료형  (*함수 포인터변수이름)(자료형1, 자료형2, ...);
```

```
void add(double*, double, double);
void subtract(double*, double, double);
   ...
   void (*pf1)(double *z, double x, double y) = add;
   void (*pf2)(double *z, double x, double y) = subtract;
   pf2 = add;
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 624쪽

변수이름이 pf인 함수 포인터를 하나 선언

- 함수 포인터 pf는 함수 add()의 주소 저장 가능
  - 함수원형이 다음인 함수의 주소를 저장
    - void 함수이름(double*, double, double);
- 주의할 점
  - (*pf)
    - 변수이름인 pf 앞에는 *이 있어야 하며 반드시 괄호를 사용
  - 만일 괄호가 없으면 함수원형
    - pf는 함수 포인터 변수가 아니라 void *를 반환하는 함수의 헤더

함수 포인터 선언

```
//잘못된 함수 포인터 선언
void *pf(double*, double, double);    //함수원형이 되어 pf는 원래 함수이름

void (*pf)(double*, double, double); //함수 포인터
pf = add; //변수 pf에 함수 add의 주소값을 대입 가능
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 624쪽

함수 포인터 변수에 대입 1/2

- 함수 포인터 변수 pf
  - add( )와 반환형과 인자목록이 같은 함수는 모두 가리킬 수 있음
  - subtract( )의 반환형과 인자목록이 add( )와 동일하다면
    - pf는 함수 subtract( )도 가리킬 수 있음
  - 문장 pf = subtract;
    - 함수 포인터에는 괄호가 없이 함수이름만으로 대입
    - 함수 add나 subtract는 주소 연산자를 함께 사용하여 &add나 &subtract로도 사용 가능
    - subtract( )와 add( )와 같이 함수호출로 대입해서는 오류가 발생

함수 포인터 변수에 대입 2/2

```
void (*pf2)(double *z, double x, double y) = add();      //오류발생
pf2 = subtract();                                        //오류발생
pf2 = add;                                               //가능
pf2 = &add;                                              //가능
pf2 = subtract;                                          //가능
pf2 = &subtract;                                         //가능
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 625쪽

포인터 개념

- 주소값을 저장하는 변수
  - 일반적으로 포인터가 가리키는 대상의 구체적인 자료형의 포인터로 사용
  - int *, double *
- 주소값이란 참조를 시작하는 주소에 불과
  - 자료형을 알아야 참조할 범위와 내용을 해석할 방법을 알 수 있음

void 포인터 개념

- void 포인터는 자료형을 무시하고 주소값만을 다루는 포인터
  - 대상에 상관없이 모든 자료형의 주소를 저장할 수 있는 만능 포인터로 사용 가능
  - void 포인터에는 일반 포인터는 물론 배열과 구조체 심지어 함수 주소도 저장 가능

void 포인터

```
char ch = 'A';
int data = 5;
double value = 34.76;

void *vp;           //void 포인터 변수 vp 선언

vp = &ch;           //ch의 주소만을 저장
vp = &data;         //data의 주소만을 저장
vp = &value;        //value의 주소만을 저장
```

[이미지] 강환수 외 2인 공저, Perfect C (3판), 인피니티북스, 2021, 630쪽

정 리 하 기

정리하기

- 함수에서 매개변수로 값을 전달하는 방식은
  값의 전달(call by value) 방식과
  주소에 의한 전달(call by address) 방식으로 나뉜다.
- 함수에서 배열을 매개변수로 사용하는 경우,
  배열의 크기도 알려줘야 한다.
- 매개변수를 수정 불가능하게 하려면 const를 사용한다.
- 함수는 포인터는 함수의 주소를 저장할 수 있는 변수로 반환형,
  인자목록의 수와 각각의 자료형이 일치해야 한다.
- 자료형 void 포인터(void *)는
  자료형에 상관없이 주소값만을 다루는 포인터로
  모든 자료형의 주소를 저장할 수 있다.

13강

파일 처리

다음시간안내