WEEK 13

함수와 포인터 활용

학습목표

I. 함수의 인자전달 방식 이해
II. 함수에서 인자로 포인터의 전달과 포인터형의 사용 이해
III. 함수 포인터 이해

학습목차

I. 제 1교시 함수의 인자전달 방식
II. 제 2교시 포인터 전달과 반환
III. 제 3교시 함수 포인터와 void 포인터

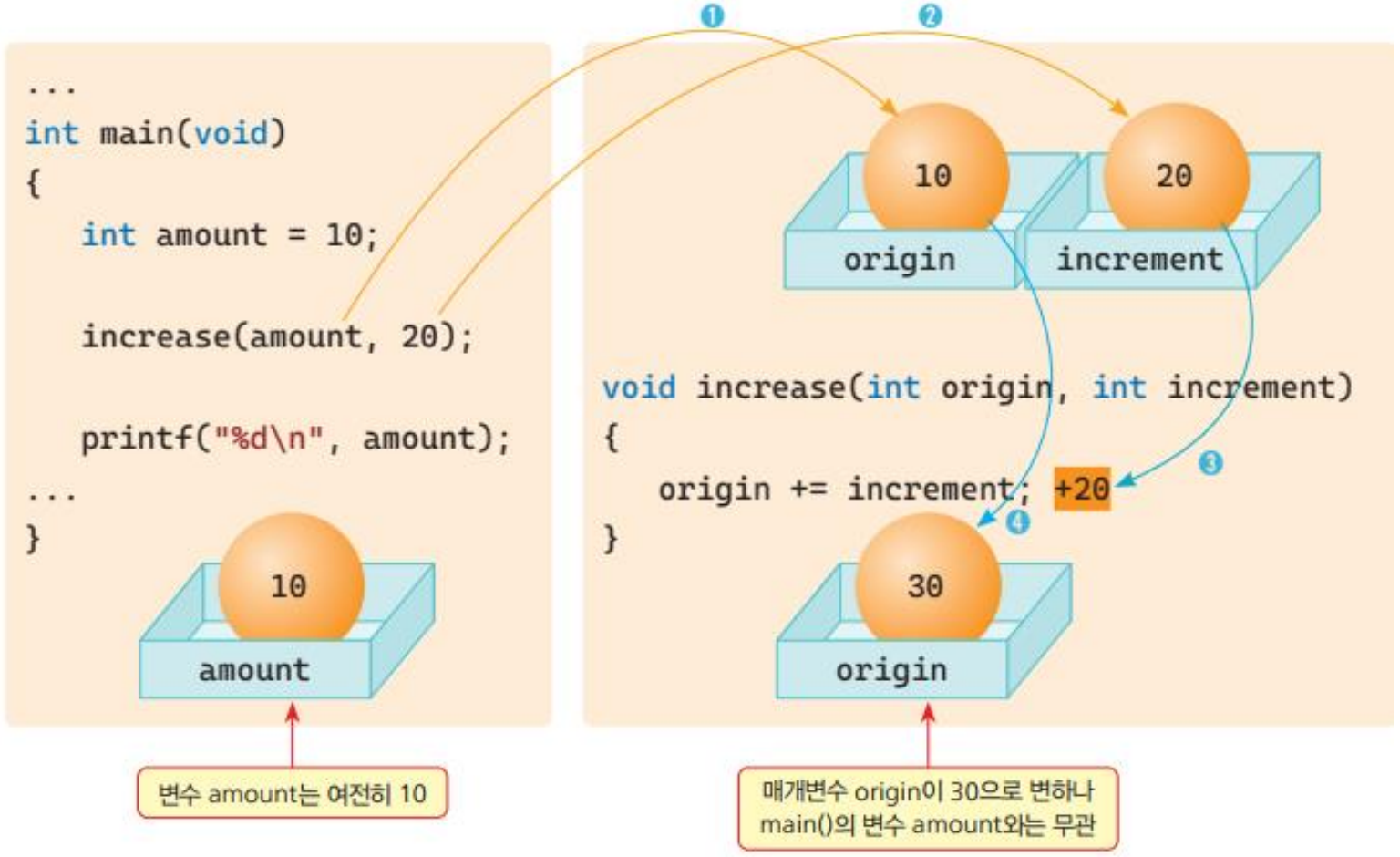
Ⅰ. 함수의 인자전달 방식

1. 값에 의한 호출과 참조에 의한 호출
2. 배열의 전달
3. 가변인자

1. 값에 의한 호출과 참조에 의한 호출

◆ 값에 의한 호출(call by value) 방식

❖ 함수 호출 시 실인자의 값이 형식인자에 복사되어 저장된다는 의미



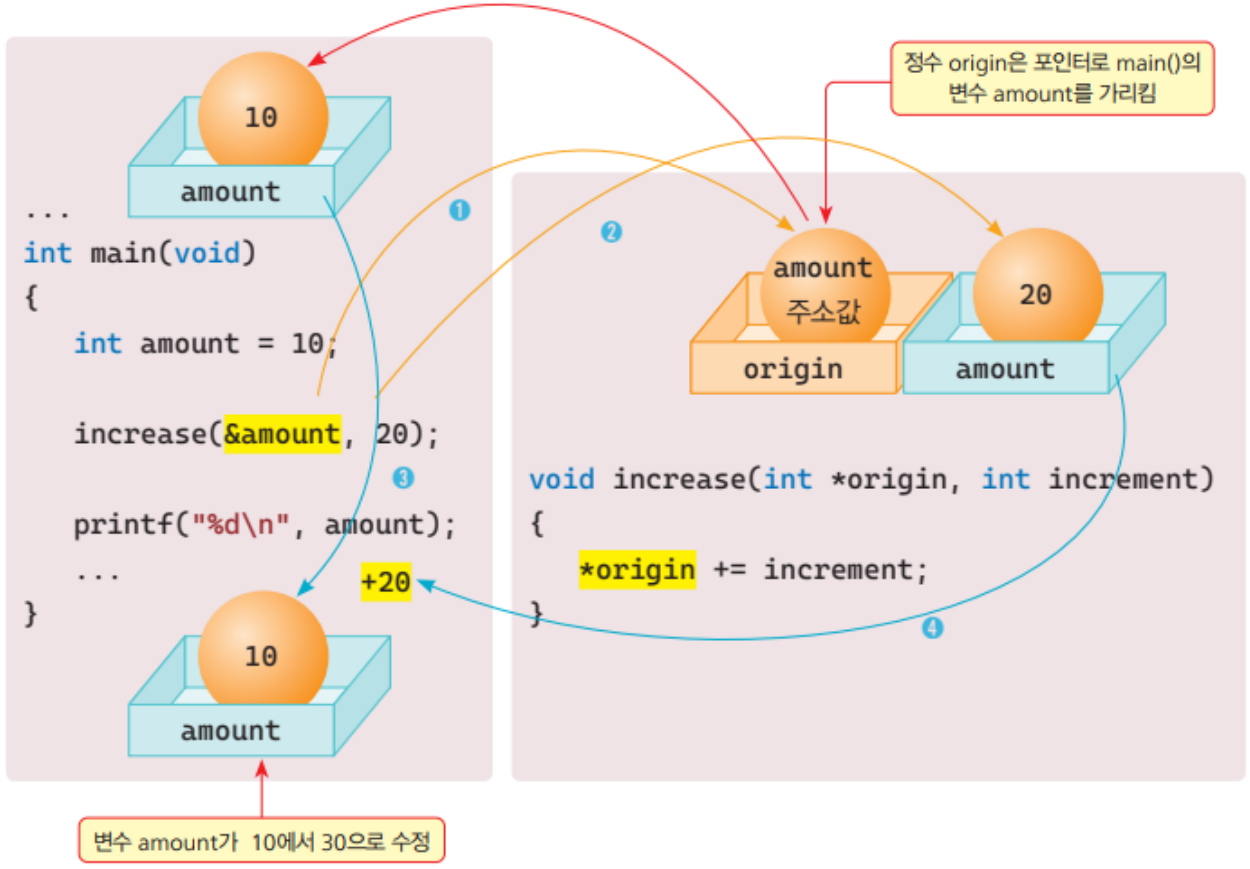
[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 값에 의한 호출과 참조에 의한 호출

◆ 주소에 의한 호출(call by address)

❖ 포인터를 매개변수로 사용

- 함수로 전달된 실인자의 주소를 이용하여 그 변수를 참조 가능



[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-1 | Prj01 | 01callby.c | 함수의 값과 주소의 전달 방식 | 난이도: ★★

```c
#include <stdio.h>

void increase(int origin, int increment);
void incbyaddress(int* origin, int increment);

int main(void)
{
   int amount = 10;
   increase(amount, 20); //amount가 20 증가하지 않음
   printf("%d\n", amount);

   amount = 10;
   incbyaddress(&amount, 20); //&amount: amount의 주소로 호출
   printf("%d\n", amount);

   return 0;
}

void increase(int origin, int increment)
{
   origin += increment;
}
void incbyaddress(int* origin, int increment)
{
   //*orogin은 origin이 가리키는 변수 자체
   *origin += increment; //그러므로 origin이 가리키는 변수 값이 20 증가
}
```

포인터 변수인 origin에 amount의 주소를 저장하고, 변수 increment에 20이 각각 저장

변수 origin이 가리키는 값은 amount, 그러므로 amount가 20 증가

결과

```
10
30
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 배열의 전달

❖ 함수의 매개변수로 배열을 전달하는 것

- 배열의 첫 원소를 참조 매개변수로 전달하는 것과 동일

❖ 배열을 매개변수로 하는 함수 sum()을 구현

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

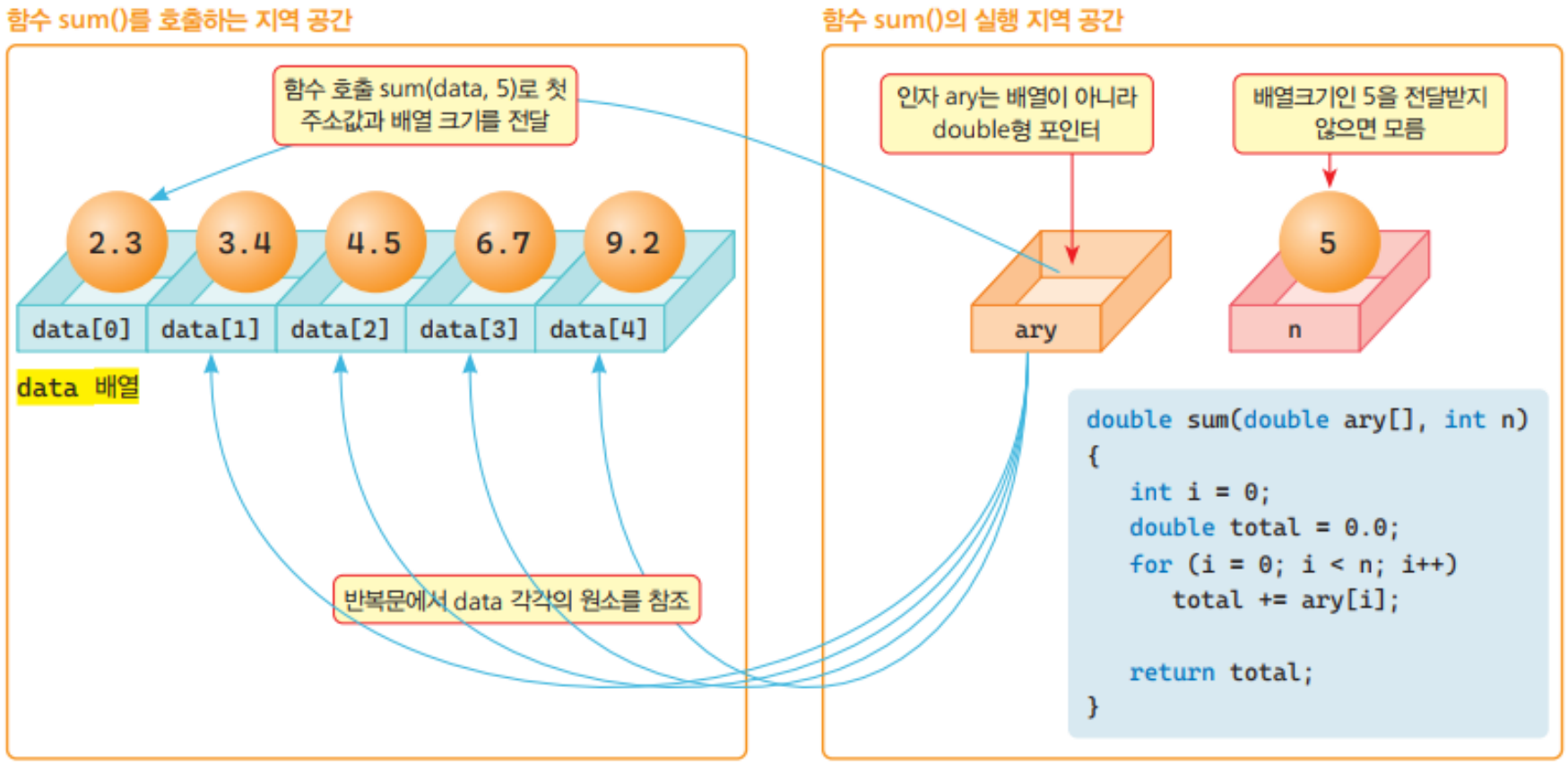
```
double sum(double ary[], int n)
{
   int i = 0;
   double total = 0.0;
   for (i = 0; i < n; i++)
      total += ary[i];

   return total;
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 배열의 전달

❖ 배열크기에 관계없이 배열 원소의 합을 구하는 함수를 만들려면

- 배열크기도 하나의 인자로 사용



[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 배열의 전달

◆ 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;
  - 배열 point를 가리키는 포인터 변수 address를 선언하여 point를 저장
- 문장 sum += *(address++)으로도 배열의 합 가능
- 배열이름 point는 주소 상수
  - sum += *(point++)는 사용 불가능
  - 증가 연산식 point++의 피연산자로 상수인 point를 사용할 수 없기 때문

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 배열의 전달

◆ 다양한 배열원소 참조 방법

```
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++);
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 배열의 전달

◆ 함수 머리에 배열을 인자로 기술하는 다양한 방법

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++);
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Prj02 02aryparam.c 함수에서 배열인자의 사용 난이도: ★★

```c
#include <stdio.h>

int sumary(int*, int); //int sumary(int ary[], int SIZE)도 가능

int main(void)
{
   int point[] = { 95, 88, 76, 54, 85, 33, 65, 78, 99, 82 };
   int aryLength = sizeof(point) / sizeof(int); //배열크기 구하기

   int* address = point; //address는 포인터 변수이며 point는 배열 상수
   int sum = 0;
   for (int i = 0; i < aryLength; i++) //메인에서 직접 배열 합 구하기
      sum += *(point + i);    //*(address++), *(address + i)도 가능
      //sum += *(point++);    //오류발생
   printf("main()에서 구한 합은 %d\n", sum);

   //함수 호출하여 합 구하기, 첫 인자 &point[0] 또는 address로도 가능
   printf("함수 sumary() 호출로 구한 합은 %d\n", sumary(point, aryLength));

   return 0;
}

int sumary(int* ary, int SIZE) //int sumary(int ary[], int SIZE)도 가능
{
   int sum = 0;
   for (int i = 0; i < SIZE; i++)
   {
      sum += *ary++; //*(ary++)와 동일
      //sum += ary[i];    //가능
      //sum += *(ary + i);    //가능
   }

   return sum;
}
```

함수원형에서 변수 이름 자체는 생략 가능

```
main()에서 구한 합은 755
함수 sumary() 호출로 구한 합은 755
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 가변인자

❖ 함수 printf() 함수원형

- 첫 인자는 char *_Format을 제외하고는 이후에 ... 표시

❖ 함수 printf()를 호출하는 경우를 살펴보면

- 출력할 인자의 수와 자료형이 결정되지 않은 채 함수를 호출
- 출력할 인자의 수와 자료형은 인자 _Format에 %d 등으로 표현

```
//함수 printf()의 함수원형
int printf(const char *_Format, ...); //...이 무엇일까?

//함수 사용 예
printf("%d%d%f", 3, 4, 3.678); //인자가 총 4개
printf("%d%d%f%f%f", 7, 9, 2.45, 3.678, 8.98); //인자가 총 5개
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 가변인자

❖ 함수에서 인자의 수와 자료형이 결정되지 않은 함수 인자 방식

- 매개변수에서 중간 이후부터 마지막에 위치한 가변 인자만 가능
  - 처음 또는 앞 부분의 매개변수는 정해져 있으나
  - 이후 매개변수 수와 각각의 자료형이 고정적이지 않고 변하는 인자
- ...으로 기술
  - 가변인자의 매개변수

```
int vatest(int n, ...);
double vasum(char *type, int n, ...);
double vafun1(char *type, ..., int n); //오류, 마지막이 고정적일 수 없음
double vafun2(...); //오류, 처음부터 고정적일 수 없음
```

❖ 함수 vatest의 함수 헤드

- void vatest(int n, ...)
  - 가변 인자인 ...의 앞 부분에는 반드시 매개변수가 int n처럼 고정적이어야 함
  - 가변인자 ... 시작 전 이전 고정 매개변수 n
  - 가변인자를 처리하는데 필요한 정보를 지정하는데 사용

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 가변인자

- ❖ 헤더파일 stdarg.h 필요
- ❖ 함수에서 가변인자 구현 과정

처리 절차	구문	설명
❶ 가변인자 선언	va_list argp;	va_list로 변수 argp을 선언
❷ 가변인자 처리 시작	va_start(va_list argp, prevarg)	va_start()는 첫 번째 인자로 va_list로 선언된 변수이름 argp과 두 번째 인자는 가변인자 앞의 고정인자 prevarg를 지정하여 가변인자 처리 시작
❸ 가변인자 얻기	type va_arg(va_list argp, type)	va_arg()는 첫 번째 인자로 va_start()로 초기화한 va_list 변수 argp를 받으며, 두 번째 인자로는 가변인자로 전달된 값의 type을 기술
❹ 가변인자 처리 종료	va_end(va_list argp)	va_list로 선언된 변수이름 argp의 가변인자 처리 종료

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 가변인자

❖ 함수 sum(int numargs, ...)

가변인자 선언
• va_list 가변인자변수;

가변인자 처리시작
• va_start(가변인자변수, 가변인자_이전_첫_고정인자)

가변인자 얻기
• va_arg(가변인자변수, 반환될_자료형)

가변인자 처리종료
• va_end(가변인자변수)

```
int sum(int numagrs, ...)
{
    va_list argp;

    //가변인자 처리 시작
    va_start(argp, numagrs);

    ...
    total += va_arg(argp, int);

    ...

    //가변인자 처리 종료
    va_end(argp);

      ...
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-3 | Prj03 | 03vararg.c | 가변인자 처리 함수의 정의와 호출 | 난이도: ★★★

```c
#include <stdio.h>
#include <stdarg.h> //가변인자를 위한 헤더 파일

double avg(int, ...); //int 이후는 가변인자 ...

int main(void)
{
   printf("평균 %.2f\n", avg(5, 1.2, 2.1, 3.6, 4.3, 5.8));
   printf("평균 %.2f\n", avg(6, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0));

   return 0;
}

//가변인자 ... 시작 전 첫 고정 매개변수는 가변인자를 처리하는데 필요한 정보를 지정
double avg(int numagrs, ...) //매개변수 numargs는 가변인자의 수를 지정
{
   double total = 0; //합이 저장될 변수

   va_list argp; //1. 가변인자 변수 선언
   va_start(argp, numagrs); //2. numargs 이후의 가변인자 처리 시작
   for (int i = 0; i < numagrs; i++) //3. 가변인자 얻기
      total += va_arg(argp, double); //지정하는 double 형으로 가변인자 하나를 반환
   va_end(argp); //4. 가변인자 처리 종료

   return total / numagrs;
}
```

가변인자 바로 앞 인자를 기술

결과

```
평균 3.40
평균 3.50
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅰ. 함수의 인자전달 방식

1교시 수업을 마치겠습니다.

Ⅱ. 포인터 전달과 반환

1. 함수 매개변수와 반환으로 포인터 사용
2. 상수를 위한 const 사용
3. 함수의 구조체 전달과 반환

1. 함수 매개변수와 반환으로 포인터 사용

❖ 주소연산자 &

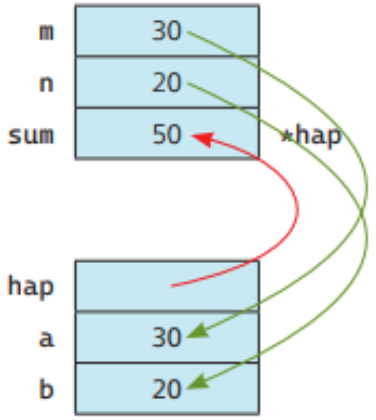
- 함수에서 매개변수를 포인터로 이용하면 결국 주소에 의한 호출
- 함수원형 void add(int *, int, int); 에서 첫 매개변수가 포인터인 int *
  - 함수 add()는 두 번째와 세 번째 인자를 합해 첫 번째 인자가 가리키는 변수에 저장 함수
  - 변수인 sum을 선언하여 주소값인 &sum을 인자로 호출

```
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

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

함수 add() 정의에서 합을 저장하는 문장



[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-4 | Prj04 | 04ptrparam.c | 함수로 포인터를 전달하는 주소에 의한 호출 | 난이도: ★★

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void add(int*, int, int);

int main(void)
{
   int m = 0, n = 0, sum = 0;

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

   return 0;
}

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

m과 n을 더한 결과가 변수 sum에 저장

결과

```
두 정수 입력: 10 20
두 정수 합: 30
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 매개변수와 반환으로 포인터 사용

❖ 주소값 반환

- 함수원형을 int * add(int *, int, int) 로 하는 함수 add()
  - 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)로 호출
  - 변수 sum에 합 a+b가 저장
  - 반환값인 포인터가 가리키는 변수인 sum을 바로 참조

```
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;
}
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-5 | Prj05 | 05ptrreturn.c | 주소값 반환 함수의 정의와 이용 | 난이도: ★★★

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

int* add(int*, int, int);
int* subtract(int*, int, int);
int* multiply(int, int);

int main(void)
{
   int m = 0, n = 0, sum = 0, diff = 0;

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

   printf("두 정수 합: %d\n", *add(&sum, m, n));
   printf("두 정수 차: %d\n", *subtract(&diff, m, n));
   printf("두 정수 곱: %d\n", *multiply(m, n));

   return 0;
}

int* add(int* psum, int a, int b)
{
   *psum = a + b;
   return psum;
}
int* subtract(int* pdiff, int a, int b)
{
   *pdiff = a - b;
   return pdiff;
}
int* multiply(int a, int b)
{
   int mult = a * b;
   return &mult;
}
```

반환값인 합이 저장된 주소이므로 *add()로 결과를 바로 출력 가능

m과 n을 곱이 저장된 함수 multiply()의 지역변수 주소값을 전달하므로 경고가 발생

warning C4172: 지역 변수 또는 임시: mult의 주소를 반환하고 있습니다.

결과

```
두 정수 입력: 50 5
두 정수 합: 55
두 정수 차: 45
두 정수 곱: 250
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 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

```
// 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;
}
```

❖ const double *a, const double *b

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

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Prj06 06constparam.c 함수 매개변수에서 const의 사용 방법과 의미 난이도: ★★

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void multiply(double*, const double*, const double*);
void devideandincrement(double*, double*, double*);

int main(void)
{
   double m = 0, n = 0, mult = 0, dev = 0;

   printf("두 실수 입력: ");
   scanf("%lf %lf", &m, &n);
   multiply(&mult, &m, &n);
   devideandincrement(&dev, &m, &n);
   printf("두 실수 곱: %.2f, 나눔: %.2f\n", mult, dev);
   printf("연산 후 두 실수: %.2f, %.2f\n", m, n);

   return 0;
}
```

```c
//매개변수 포인터 a, b가 가리키는 변수의 내용을 곱해 result가 가리키는 변수에 저장
void multiply(double* result, const double* a, const double* b)
{
   *result = *a * *b;
   //*a = *a + 1; //오류발생
   //*b = *b + 1; //오류발생
}

//매개변수 포인터 a, b가 가리키는 변수의 내용을 나누어 result가 가리키는 변수에 저장한 후
//a, b가 가리키는 변수의 내용을 모두 1 증가시킴
void devideandincrement(double* result, double* a, double* b)
{
   *result = *a / *b;
   ++* a;  //++(*a)이므로 a가 가리키는 변수의 값을 1 증가
   (*b)++; //b가 가리키는 변수의 값을 1 증가, *b++와는 다름
}
```

매개변수 포인터 a, b가 가리키는 변수의 내용은 수정하지 못함

const인 인자 *a와 *b로는 수정할 수 없으므로 다음과 같은 문법 오류가 발생 "error C2166: l-value가 const 개체를 지정합니다."

const가 아닌 포인터 인자 *result, *a와 *b는 모두 수정 가능

```
두 실수 입력: 12.5 4.5
두 실수 곱: 56.25, 나눔: 2.78
연산 후 두 실수: 13.50, 5.50
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 상수를 위한 const 사용

TIP 참조 변수의 상수와 포인터의 상수

앞의 11장 포인터 기초에서 학습했듯이 const의 위치가 다음과 같이 *와 포인터 변수 사이라면 포인터 변수 자체가 상수라는 의미이다.

- 키워드 const가 int*와 변수 pi 사이에 나오는 선언은 포인터 pi에 저장되는 초기 주소값을 더 이상 수정할 수 없도록 하는 상수이다. 즉 포인터 변수 pi 자체를 상수로 만드는 방법으로 pi를 l-value로 사용할 수 없다.

```
int i = 10, j = 20;
int* const pi = &i;
pi = &j; //오류 발생
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 함수의 구조체 전달과 반환

```
struct complex
{
    double real; //실수
    double img;  //허수
};
typedef struct complex complex;
```

자료형 struct complex

자료형 complex

자료형 struct complex와 complex 모두 복소수 자료형으로 사용 가능

- 복소수의 합: (a + bi) + (c + di) = (a + b) + (c + d)i
- 복소수의 곱: (a + bi) * (c + di) = (ac – db) + (ad + bc)i
- (a + bi)의 켤레 복소수: (a – bi)
- (a – bi)의 켤레 복소수: (a + bi)

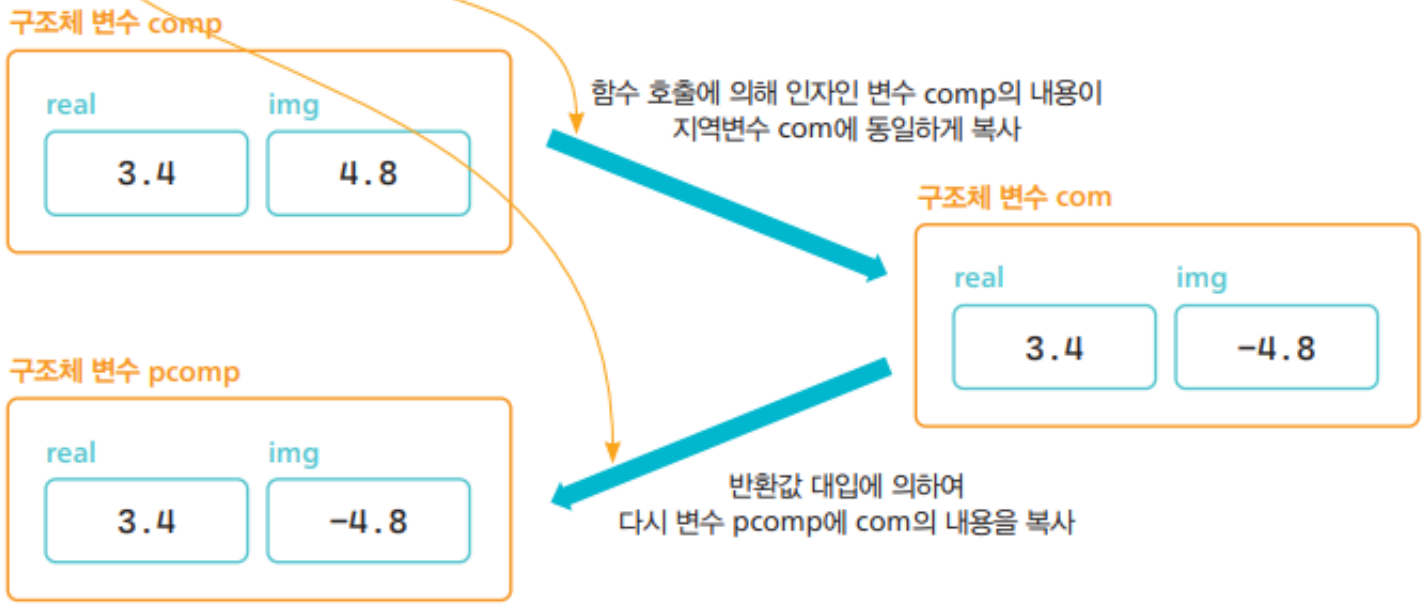
[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 함수의 구조체 전달과 반환

❖ 인자인 복소수의 켤레 복소수(pair complex number)를 구하여 반환하는 함수

```
complex comp = { 3.4, 4.8 };
complex pcomp;

pcomp = pcomplexvalue(comp);
```

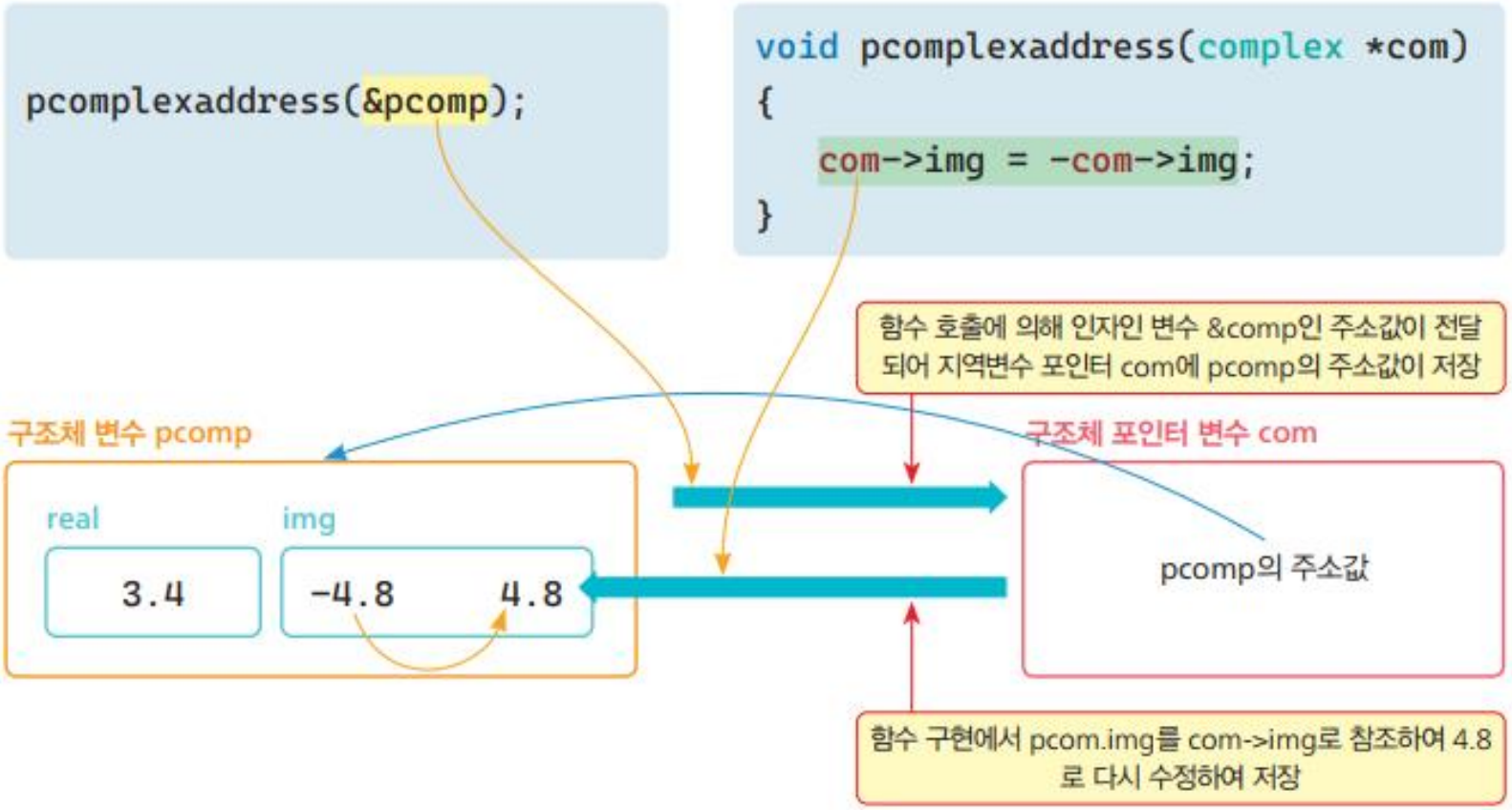
```
complex pcomplexvalue(complex com)
{
    com.img = -com.img;
    return com;
}
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. 함수의 구조체 전달과 반환

◆ 주소에 의한 호출(call by address) 방식으로 수정



[출처] 강환수 외, Perfect C 3판, 인피니티북스

Prj07 07complexnum.c 구조체를 사용하여 복소수를 표현, 함수의 인자와 반환형으로 사용 난이도: ★★

```c
#include <stdio.h>

struct complex //복소수를 위한 구조체
{
   double real; //실수
   double img;  //허수
};
typedef struct complex complex; //complex를 자료형으로 정의

void printcomplex(complex com);
complex pcomplexvalue(complex com);
void pcomplexaddress(complex* com);

int main(void)
{
   complex comp = { 5.8, 7.2 };
   complex pcomp;

   printcomplex(comp);
   pcomp = pcomplexvalue(comp);
   printcomplex(pcomp);
   pcomplexaddress(&pcomp);
   printcomplex(pcomp);

   return 0;
}

//구조체 자체를 인자로 사용
void printcomplex(complex com)
{
   printf("복소수 = %5.1f + %5.1fi \n", com.real, com.img);
}
```

구조체 변수 comp의 켤레 복소수를 변수 pcomp에 저장하기 위해 변수 comp를 인자로 함수 pcomplexvalue(comp) 호출하여 그 결과를 바로 변수 pcomp에 대입, 함수 호출에서 인자 대입과 반환 값 대입 등 구조체 자체의 대입이 두 번 필요하므로 시간이 소요됨

구조체 변수 pcomp의 켤레 복소수를 다시 변수 pcomp에 반영하기 위해 &pcomp를 인자로 함수 pcomplexaddress(&pcomp) 호출하면 바로 그 결과가 변수 pcomp에 반영, 함수 호출에서 구조체의 주소값만 대입되므로 위 pcomplexvalue 보다 빠르게 처리 가능

```c
//구조체 자체를 인자로 사용하여 처리된 구조체를 다시 반환
complex pcomplexvalue(complex com)
{
   com.img = -com.img;
   return com; //구조체를 반환
}

//구조체 포인터를 인자로 사용
void pcomplexaddress(complex* com)
{
   com->img = -com->img;
}
```

```
복소수 =   5.8 +   7.2i
복소수 =   5.8 +  -7.2i
복소수 =   5.8 +   7.2i
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅱ. 포인터 전달과 반환

2교시 수업을 마치겠습니다.

Ⅲ. 함수 포인터와 void 포인터

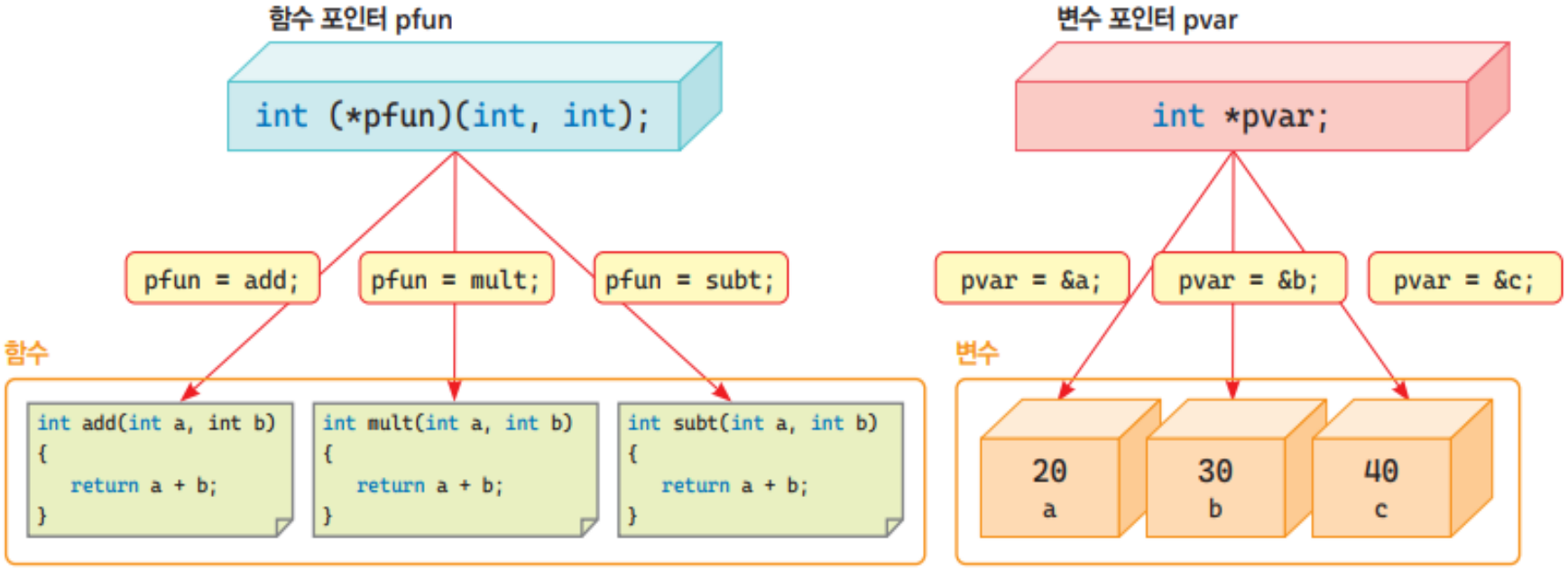
1. 함수 포인터
2. 함수 포인터 배열
3. void 포인터

1. 함수 포인터

❖ 하나의 함수 이름으로 필요에 따라 여러 함수를 사용하면 편리

❖ 함수 포인터 pfun

- 함수 add()와 mult() 그리고 subt()로도 사용 가능



[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 포인터

- 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

```
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;
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 포인터

❖ 변수이름이 pf인 함수 포인터를 하나 선언

- 함수 포인터 pf는 함수 add()의 주소 저장 가능
  - 함수원형이 void add(double*, double, double);인 함수의 주소를 저장
  - 함수원형에서 반환형인 void와 인자목록인 (double *, double, double) 정보 필요
- 여기서 주의할 점
  - (*pf)와 같이 변수이름인 pf 앞에는 *이 있어야 하며 반드시 괄호를 사용
  - 만일 괄호가 없으면 함수원형
  - pf는 함수 포인터 변수가 아니라 void *를 반환하는 함수이름

```
//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
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 포인터

❖ 물론 앞의 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

```
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
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 포인터

❖ 함수 add()의 구현

- 함수 add()에서 x + y의 결과를 반환하지 않고 포인터 변수 z에 저장
  - 인자를 포인터 변수로 사용하면 함수 내부에서 수정한 값이 그대로 실인자로 반영
- 문장 pf = add;
  - 함수 포인터 변수인 pf에 함수 add()의 주소값이 저장
  - 변수 pf를 이용하여 add() 함수를 호출 가능
- 포인터 변수 pf를 이용한 함수 add()의 호출방법
  - add() 호출과 동일
  - 즉, pf(&result, m, n); 또는 (*pf)(&result, m, n)로 add(&result, m, n) 호출을 대체
  - 문장이 실행되면 변수 result에는 m + n의 결과가 저장

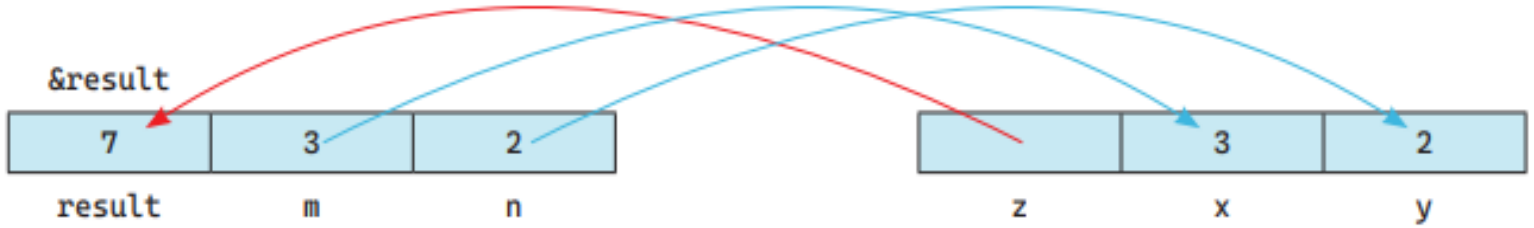
[출처] 강환수 외, Perfect C 3판, 인피니티북스

1. 함수 포인터

❖ 함수 add()의 구현

```
double m, n, result = 0;
void (*pf)(double*, double, double);
....
pf = add;
pf(&result, m, n); //add(&result, m, n);
//(*pf)(&result, m, n); //이것도 사용 가능
```

```
void add(double *z, double x, double y)
{
   *z = x + y;
}
```



[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-8 | Prj08 | 08ptrfunction.c | 함수 주소를 저장하는 함수 포인터의 선언과 사용 | 난이도: ★★

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void add(double*, double, double);
void subtract(double*, double, double);

int main(void)
{
   void (*pf)(double*, double, double) = NULL; //함수 포인터 pf를 선언

   double m, n, result = 0;
   printf("연산 +, -를 수행할 실수 2개를 입력하세요. >> ");
   scanf("%lf %lf", &m, &n);

   pf = add; //add() 함수를 함수 포인터 pf에 저장
   pf(&result, m, n);  //add(&result, m, n);
   printf("\n더하기 수행: %lf + %lf == %lf\n", m, n, result);
   printf("%p %p\n", pf, add);

   pf = subtract; //subtract() 함수를 함수 포인터 pf에 저장
   pf(&result, m, n);  //subtract(&result, m, n);
   printf("  빼기 수행: %lf - %lf == %lf\n", m, n, result);
   printf("%p %p\n", pf, subtract);

   return 0;
}

// x + y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void add(double* z, double x, double y)
{
   *z = x + y;
}
// x - y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void subtract(double* z, double x, double y)
{
   *z = x - y;
}
```

함수 포인터 pf를 선언, 반환 값은 없으며(void), 인자의 유형은 double *, double, double 이라는 것으로 선언하면서 초기값으로 0번지 주소인 NULL을 저장

add() 함수의 주소값을 함수 포인터 pf에 저장하는 문장

함수 포인터 pf로 함수를 호출하는 방법은 add()와 동

pf와 add 모두 함수의 주소값 출력

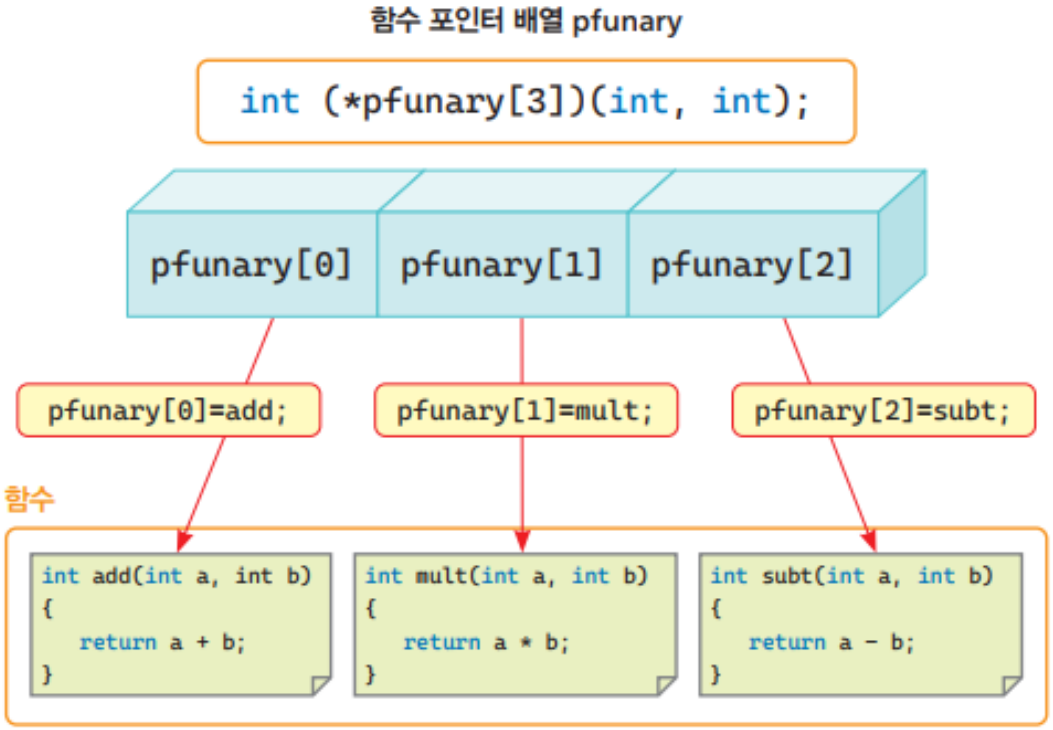
결과

```
연산 +, -를 수행할 실수 2개를 입력하세요. >> 5.3 1.2

더하기 수행: 5.300000 + 1.200000 == 6.500000
00007FF61D5B105F 00007FF61D5B105F
  빼기 수행: 5.300000 - 1.200000 == 4.100000
00007FF61D5B117C 00007FF61D5B117C
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 함수 포인터 배열

- 원소로 여러 개의 함수 포인터를 선언하는 함수 포인터 배열
  - 크기가 3인 함수 포인터 배열 pfunary는 문장 int (*pfunary[3])(int, int); 으로 선언
- 배열 pfunary의 각 원소가 가리키는 함수
  - 반환값이 int이고 인자목록이 (int, int)



[출처] 강환수 외, Perfect C 3판, 인피니티북스

2. 함수 포인터 배열

함수 포인터 배열 선언

```
반환자료형  (*배열이름[배열크기])(자료형1 매개변수이름1, 자료형2 매개변수이름2, ...);
반환자료형  (*배열이름[배열크기])(자료형1, 자료형2, ...);
```

```
void add(double*, double, double);
void subtract(double*, double, double);
void multiply(double*, double, double);
void devide(double*, double, double);
   ...
   void (*fpary[4])(double*, double, double);
```

```
void (*fpary[4])(double*, double, double);
fpary[0] = add;
fpary[1] = subtract;
fpary[2] = multiply;
fpary[3] = devide;
```

배열의 선언과 초기화 문장으로 간단히 처리

```
void (*fpary[4])(double*, double, double) = {add, subtract, multiply, devide};
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅲ. 함수 포인터와 void 포인터

Prj09 **09aryptrf.c** 여러 함수 주소를 저장하는 함수

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void add(double*, double, double);
void subtract(double*, double, double);
void multiply(double*, double, double);
void devide(double*, double, double);

int main(void)
{
   char op[4] = { '+', '-', '*', '/' };
   //함수 포인터 선언하면서 초기화 과정
   void (*fpary[4])(double*, double, double) =
                                   { add, subtract, multiply, devide };

   double m, n, result;
   printf("사칙연산을 수행할 실수 2개를 입력하세요. >> ");
   scanf("%lf %lf", &m, &n);
   //사칙연산을 배열의 첨자를 이용하여 수행
   for (int i = 0; i < 4; i++)
   {
      fpary[i](&result, m, n);
      printf("%.2lf %c %.2lf == %.2lf\n", m, op[i], n, result);
   }

   return 0;
}

// x + y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void add(double* z, double x, double y)
{
   *z = x + y;
}
// x - y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void subtract(double* z, double x, double y)
{
   *z = x - y;
}
// x * y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void multiply(double* z, double x, double y)
{
   *z = x * y;
}
// x / y 연산 결과를 z가 가리키는 변수에 저장하는 함수
void devide(double* z, double x, double y)
{
   *z = x / y;
}
```

함수 포인터 배열 fpary[4]를 선언, 배열 크기는 4이며, 각각의 포인터는 반환 값이 없으며(void), 인자의 유형은 double *, double, double인 함수의 주소를 저장할 수 있으며, 초기값으로 함수 add, subtract, multiply, devide 의 주소를 각각 저장

제어변수 i에 따라 fpary[0]는 add()를 호출, fpary[1]은 subtract()를 호출, fpary[2]는 multiply()를 호출, fpary[3]은 devide()를 각각 호출할 때 인자는 모두 &result, m, n으로 동일

```
사칙연산을 수행할 실수 2개를 입력하세요. >> 50.56 3.4
50.56 + 3.40 == 53.96
50.56 - 3.40 == 47.16
50.56 * 3.40 == 171.90
50.56 / 3.40 == 14.87
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. void 포인터

❖ 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

```
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
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

3. void 포인터

- 모든 주소를 저장 가능하지만 가리키는 변수를 참조하거나 수정이 불가능
- 주소값으로 변수를 참조하려면 결국 자료형으로 참조범위를 알아야 하는데 void 포인터는 이러한 정보가 전혀 없이 주소 만을 담는 변수에 불과하기 때문
- void 포인터는 자료형 정보는 없이 임시로 주소 만을 저장하는 포인터
  - 그러므로 실제 void 포인터로 변수를 참조하기 위해서는 자료형 변환이 필요

```
int m = 10;      double x = 3.98;

void *p = &m;
int n = *(int *)p; //int * 로 변환
n = *p; //오류
          오류: "void" 형식의 값을 "int" 형식의 엔터티에 할당할 수 없습니다.
p = &x;
int y= *(double *)p; //double * 로 변환
y = *p; //오류
          오류: "void" 형식의 값을 "int" 형식의 엔터티에 할당할 수 없습니다.
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

실습예제 14-10 | Prj10 | 10voidptr.c | 포인터 void *의 선언과 활용 | 난이도: ★★

```c
#include <stdio.h>

void myprint(void)
{
    printf("void 포인터 신기하네요!\n");
}

int main(void)
{
    int m = 10;
    double d = 3.98;
    char str[][20] = { { "C 언어," }, { "재미있네요!" } };

    void* p = &m; //m의 주소만을 저장
    printf("%p ", p); //주소값 출력
    //printf("%d\n", *p); //컴파일 오류 발생
    printf("%d\n", *(int*) p); //int * 로 변환

    p = &d;
    printf("%p ", p); //주소값 출력
    printf("%.2f\n", *(double*) p); //double * 로 변환

    p = myprint;
    ((void(*)(void)) p)(); //함수 포인터인 void(*)(void) 로 변환하여 호출 ()

    p = str;
    //열이 20인 2차원 배열로 변환하여 1행과 1행의 문자열 출력
    printf("%s %s\n", (char(*)[20])p, (char(*)[20])p + 1);
    printf("%s %s\n", str, str + 1);

    return 0;
}
```

포인터 void * 는 *p로 바로 m을 참조할 수 없음

int 포인터인 m의 값을 p로 참조하기 위해 형변환 연산자 사용하여 *(int *)p로 참조

double 포인터인 d의 값을 p로 참조하기 위해 형변환 연산자 사용하여 *(double *)p로 참조

함수 myprint()의 함수를 호출하기 위해 p로 형변환 연산자 사용하여 ((void(*)(void)) p)()로 호출

char의 2차원배열 str에서 첫 번째와 두 번째 행을 출력하기 위해 p로 형변환 연산자 사용하여 각각 (char(*)[20])p, (char(*)[20])p + 1로 참조

결과

```
0000006A652FFB24 10
0000006A652FFB48 3.98
void 포인터 신기하네요!
C 언어, 재미있네요!
C 언어, 재미있네요!
```

[출처] 강환수 외, Perfect C 3판, 인피니티북스

Ⅲ. 함수 포인터와 void 포인터

3교시 수업을 마치겠습니다.