

C++ 언어 강의자료

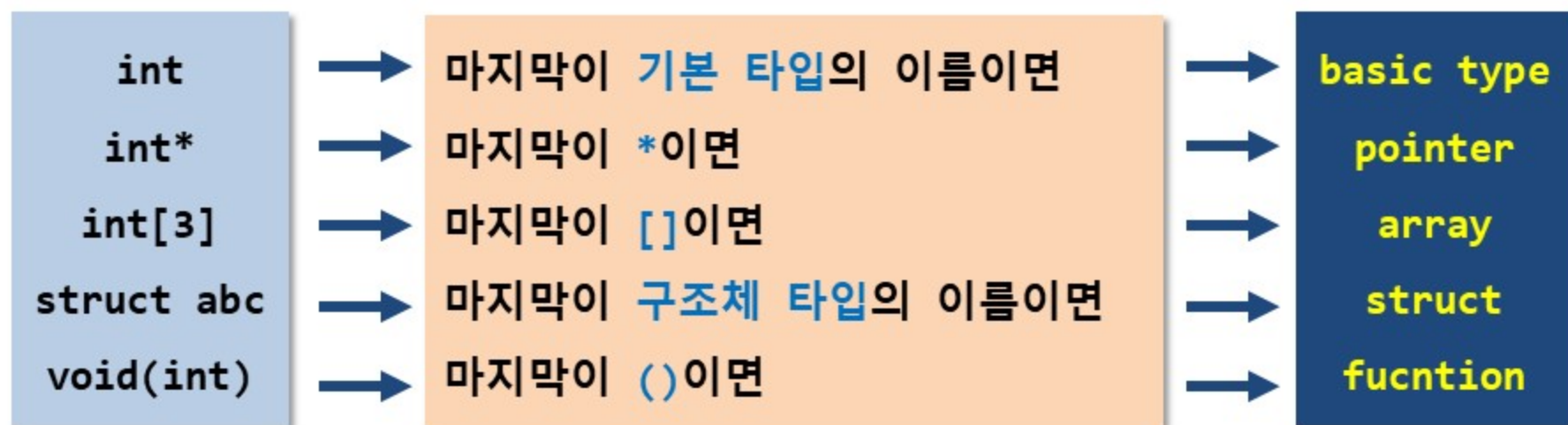
문정욱



기초이론 3: C언어 포인터 PART3(개념적 타입)

개념적 타입의 분류

타입분류 및 분류방법



개념적 타입의 분류

예제 1

<code>int*</code>	→	pointer
<code>int[3]</code>	→	array
<code>int*[3]*</code>	→	pointer
<code>int(void)*[3]*[4]</code>	→	array
<code>double*[3][4]</code>	→	array
<code>struct abc</code>	→	struct
<code>struct abc*</code>	→	pointer
<code>struct abc[3]</code>	→	array
<code>float(int)</code>	→	function
<code>float(int)*</code>	→	pointer
<code>float(int)*[3]</code>	→	array

개념적 타입의 분류

예제 2

<code>double*</code>	→	pointer	<code>char[3]*[5]</code>	→	array
<code>char[3]*</code>	→	pointer	<code>struct abc*[5]</code>	→	array
<code>struct abc*</code>	→	pointer	<code>int(char,int)*[5]</code>	→	array
<code>int(char,int)*</code>	→	pointer	<code>int(char,int)[4]</code>	→	error
<code>short**</code>	→	pointer	<code>int struct abc int[4]</code>	→	error
<code>short[5]</code>	→	array	<code>int struct abc</code>	→	error
<code>double*[5]</code>	→	array	<code>struct abc struct def</code>	→	error
<code>int[3][5]</code>	→	array	<code>char* struct abc</code>	→	error
<code>struct abc[5]</code>	→	array	<code>int(int) struct abc</code>	→	error
<code>int(char,int)[5]</code>	→	array			

개념적 타입 분류

■ 주목해야 할 점 (1/2)

- 포인터 타입일 경우 **참조 타입**에 주목해야 한다.
- * 앞쪽에 있는 내용이 참조타입에 해당한다.



```

RT( char* ) = char
RT( int*[3]* ) = int*[3]
RT( struct abc* ) = struct abc
RT( double** ) = double*
RT( float(int)* ) = float(int)
  
```

1. 포인터는 주소를 저장하여 메모리를 가리킨다.

2. 변수는 메모리 공간이므로 포인터는 모든 종류의 변수를 가리킬 수 있다.

3. 함수는 메모리 공간이므로 포인터는 모든 종류의 함수를 가리킬 수 있다.

개념적 타입 분류

■ 주목해야 할 점 (2/2)

- 배열 타입일 경우 **요소 타입**에 주목해야 한다.
- `[]` 앞쪽에 있는 내용이 요소 타입에 해당한다.

배열 타입	[] 제거	참조 타입
<code>char[4]</code>	→	<code>char</code>
<code>int*[3][4]</code>	→	<code>int*[3]</code>
<code>struct abc[4]</code>	→	<code>struct abc</code>
<code>double*[4]</code>	→	<code>double*</code>
<code>float(int)[4]</code>	→	error

1. 배열은 변수를 구성요소로 가진다.

```

ET( char[4] ) = char
ET( int*[3][4] ) = int*[3]
ET( struct abc[4] ) = struct abc
ET( double*[4] ) = double*
ET( float(int)[4] ) = none
  
```

2. 모든 종류의 변수는 배열의 구성요소가 될 수 있다.

3. 함수는 변수가 아니므로 배열의 구성요소가 될 수 없다.

개념적 타입 → C언어 타입

예제

괄호연산자를 뒤로 넘기는 순서

```

      ⑤      ④      ③      ②      ①
struct abc*(int,float)*[3]**(void)*[2][4]*    p;
struct abc*(int,float)*[3]**(void)*[2]    (*p)[4];
struct abc*(int,float)*[3]**(void)*    (*p)[4][2];
struct abc*(int,float)*[3]**    ((*p)[4][2])(void);
struct abc*(int,float)*    (**(*p)[4][2])(void))[3];
struct abc*    ((*(**(*p)[4][2])(void))[3])(int,float);
  
```

[4] 뒤에
*이 있으므로
뒤쪽 내용 전체를
()로 묶음.

(void) 뒤에
*이 있으므로
뒤쪽 내용 전체를
()로 묶음.

(int,float) 뒤에
*이 있으므로
뒤쪽 내용 전체를
()로 묶음.

[3] 뒤에
**이 있으므로
뒤쪽 내용 전체를
()로 묶음.

(주의)

- 괄호연산자에는 배열연산자와 함수연산자가 있다.
- 파란색 괄호 `()`, `[]`는 괄호연산자이다.
- 녹색 괄호 `()`는 우선기호이다.(연산자가 아님)
- 괄호연산자를 뒤로 넘기기 전 바로 뒤에 *이 있으면 바로 뒤 내용을 모두 우선기호(소괄호)로 묶은 후 괄호연산자를 뒤로 넘긴다.

개념적 타입 → C언어 타입

- 개념적 타입을 C언어 타입으로 전환하는 원칙
 - 괄호연산자(배열연산자, 함수연산자)를 뒤로 넘긴다.
 - 괄호연산자는 뒤에 있는 것부터 뒤로 넘긴다.
 - 괄호연산자를 뒤로 넘기기 전 바로 뒤에 *이 있으면 바로 뒤 내용을 모두 우선기호(소괄호)로 묶은 후 괄호연산자를 뒤로 넘긴다.
 - 함수연산자 안에 개념적 타입이 나타난다면 이는 차후 나중에 같은 방법으로 전환한다.
 - 완성된 최종 개념적 타입에서는 우선기호가 나타나지 않으므로 소괄호는 모두 함수연산자이다.

C언어 타입 → 개념적 타입

예제

괄호연산자를 앞으로 넘기는 순서



```

struct abc*      ⑤ ④ ③ ② ①
                 (*(**(*(*p)[4][2])(void))[3])(int,float);
struct abc*(int,float)*  (**(*(*p)[4][2])(void))[3];
struct abc*(int,float)*[3]**  (*(*p)[4][2])(void);
struct abc*(int,float)*[3]**(void)*  (*p)[4][2];
struct abc*(int,float)*[3]**(void)*[2]  (*p)[4];
struct abc*(int,float)*[3]**(void)*[2][4]*  p;
  
```

(int,float) 앞에
()가 있으므로
앞으로 이동 후
()를 삭제함.

[3] 앞에
()가 있으므로
앞으로 이동 후
()를 삭제함.

[4] 앞에
()가 있으므로
앞으로 이동 후
()를 삭제함.

(void) 앞에
()가 있으므로
앞으로 이동 후
()를 삭제함.

(주의)

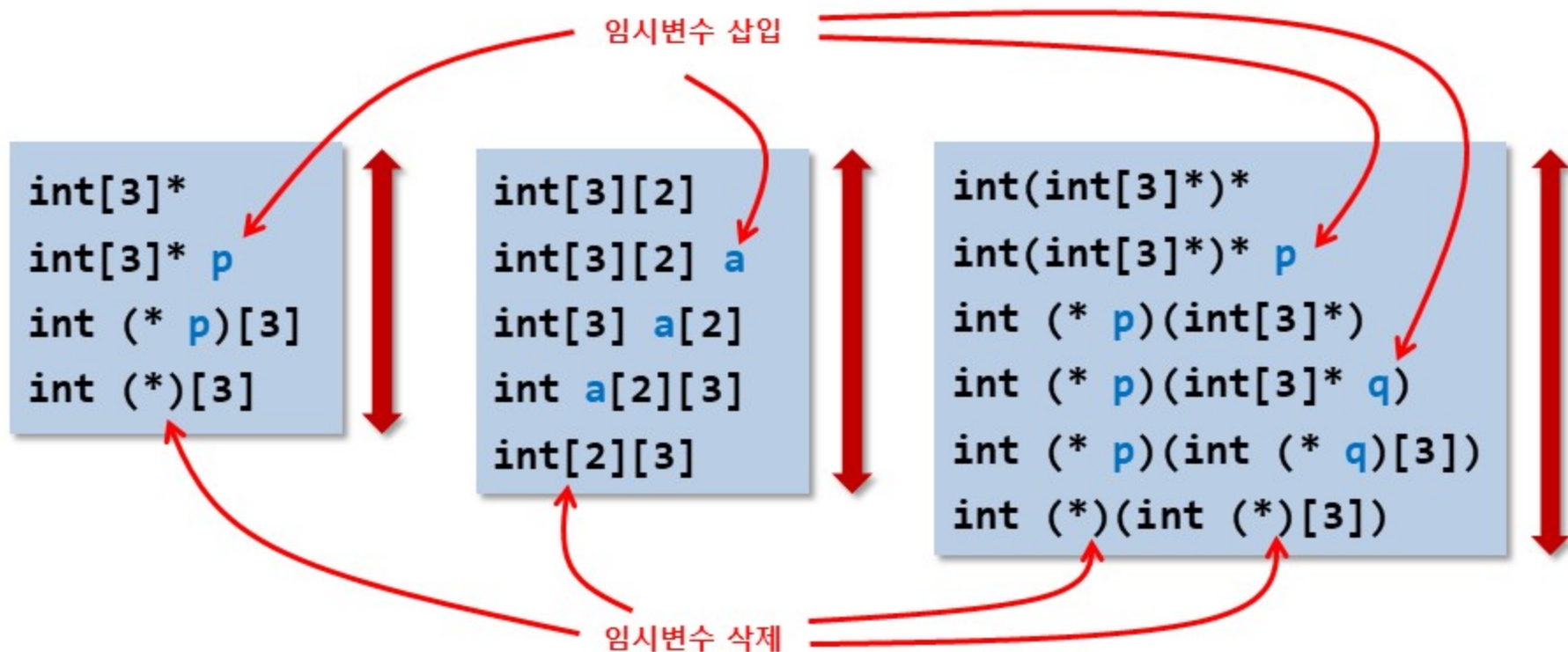
- 괄호연산자에는 배열연산자와 함수연산자가 있다.
- 파란색 괄호 (), [] 는 괄호연산자이다.
- 녹색 괄호 () 는 우선기호이다. (연산자가 아님)
- 괄호연산자 바로 앞에 있는 소괄호는 우선기호이므로 괄호연산자를 앞으로 넘긴 후 그 우선기호를 삭제한다.

C언어 타입 → 개념적 타입

- C언어 타입을 개념적 타입으로 전환하는 원칙
 - 괄호연산자(배열연산자, 함수연산자)를 앞으로 넘긴다.
 - 괄호연산자는 뒤에 있는 것부터 앞으로 넘긴다.
 - 괄호연산자 바로 앞에 있는 소괄호는 우선기호이므로 괄호연산자를 앞으로 넘긴 후 그 우선기호를 삭제한다.
 - 함수연산자 안에 C언어 타입이 나타난다면 이는 차후 나중에 같은 방법으로 전환한다.
 - C언어 타입에서는 우선기호가 마지막에 나타나지 않으므로 마지막 괄호는 괄호연산자이며 그 바로 앞에는 소괄호가 있다면 그것은 반드시 우선기호이다.

개념적 타입 ↔ C언어 타입

- 변수가 없을 때 타입의 전환: 임시변수의 삽입
 - 필요한 부분에 **임시로 변수를 삽입 후**
최종 타입이 나오면
삽입한 변수를 **다시 삭제**하면 된다.



개념적 타입 → C언어 타입

연습

```

int*[3] a;
int[3][2] a;
int[3]* p;
int(int,int)*[2] a;
int(int,int)** p;
int(int,int)*[2]* p;
int(const void*,const void*)* p;

```



```

int* a[3];
int a[2][3];
int (* p)[3];
int (* a[2])(int,int);
int (** p)(int,int);
int ((* p)[2])(int,int);
int (* p)(const void*,const void*);

```



```

int[3]*(int,int[3]*) f;
int[3]*(int,int[3]*)* p;
void(int)*(int,void(int)*) f;
void(int)*(int,void(int)*)* p;

```



```

int (* f(int,int (*)[3]))[3];
int ((* p)(int,int (*)[3]))[3];
void (* f(int,void (*) (int)))(int);
void ((* p)(int,void (*) (int)))(int);

```


C언어 타입 → 개념적 타입

연습

```

        int* a[3];
        int a[2][3];
        int (* p)[3];
        int (* a[2])(int,int);
        int (** p)(int,int);
        int ((* p)[2])(int,int);
        int (* p)(const void*,const void*);
    
```



```

int*[3] a;
int[3][2] a;
int[3]* p;
int(int,int)*[2] a;
int(int,int)** p;
int(int,int)*[2]* p;
int(const void*,const void*)* p;
    
```

```

        int (* f(int,int (*)(*)[3]))[3];
        int ((* p)(int,int (*)(*)[3]))[3];
        void (* f(int,void (*)(int)))(int);
        void ((* p)(int,void (*)(int)))(int);
    
```



```

int[3]*(int,int[3]*) f;
int[3]*(int,int[3]*)* p;
void(int)*(int,void(int)*) f;
void(int)*(int,void(int)*)* p;
    
```

타입의 재정의

- typedef
 - 기존 타입을 다른 이름으로 대체할 때 사용
- 기본 타입의 재정의
 - 새 타입의 이름은 **한 개의 단어**로 표현되어야 한다.
 - 기존 타입은 **한 개 이상의 단어**로 표현될 수 있다.

```
typedef <old_type> <new_type>;
```

↑
기존 타입

↑
새 타입

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
typedef int INT ;
typedef long long LLONG ;
typedef long long int llong ;
```

Type
declaration

```
INT a;
LLONG b;
llong c;
```

Variable
declaration

```
return 0;
```

```
}
```

타입의 재정의

해석 방법 1: 타입 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int INT ;
```

```
    typedef long long LLONG ;
```

```
    typedef long long int llong ;
```

```
    INT a;  
    LLONG b;  
    llong c;
```

```
    return 0;
```

```
}
```

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int INT ;
```

```
    typedef long long LLONG ;
```

```
    typedef long long int llong ;
```

```
    int a;  
    long long b;  
    long long int c;
```

```
    return 0;
```

```
}
```

타입의 재정의

해석 방법 2: 변수 대입법 (추천)

```
#include <stdio.h>

int main(void)
{
    typedef int INT ;
    typedef long long LLONG ;
    typedef long long int llong ;

    INT a;
    LLONG b;
    llong c;


    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    typedef int a ;
    typedef long long b ;
    typedef long long int c ;

    INT a;
    LLONG b;
    llong c;

    return 0;
}
```



타입의 재정의

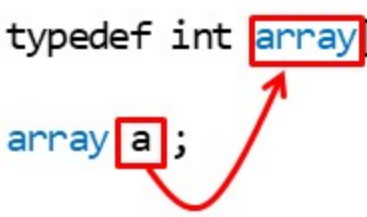
- 배열 타입의 재정의
 - 배열을 선언하듯이 재정의 기술하고 앞에 typedef 키워드를 추가한다.
 - 새 타입의 이름이 배열 이름인 듯 취급하면 됨.
 - 배열 타입을 사용하여 변수를 선언할 때는 기본 타입의 변수를 선언하는 것과 같은 방식 사용

```
#include <stdio.h>

int main(void)
{
    typedef int array[3];

    array a;

    return 0;
}
```



타입의 재정의

해석 방법 2: 변수 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int array[3];
```

```
    array a;
```

```
    return 0;
```

```
}
```

배열의 경우
변수 대입법으로
해석해야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int a[3];
```

```
    array a;
```

```
    return 0;
```

```
}
```

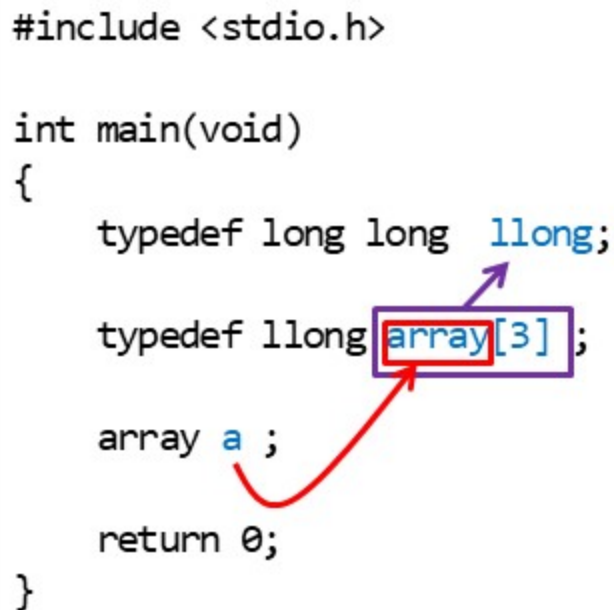
타입의 재정의

■ 이중 재정의

- 타입 재정의의 활용한 재정의
- 이중 재정의 뒷부분을 대입한다.

```
#include <stdio.h>

int main(void)
{
    typedef long long llong;
    typedef llong array[3];
    array a ;
    return 0;
}
```



타입의 재정의

해석 방법 2: 변수 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef long long  llong;
```

```
    typedef llong array[3] ;
```

```
    array a ;
```

```
    return 0;
```

```
}
```

배열의 경우
변수 대입법으로
해석해야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef long long  llong;
```

```
    typedef llong a[3] ;
```

```
    array a ;
```

```
    return 0;
```

```
}
```

타입의 재정의

해석 방법 2: 변수 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef long long  llong;
```

```
    typedef llong a[3] ;
```

```
    array a ;
```

```
    return 0;
```

```
}
```

배열의 경우
변수 대입법으로
해석해야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef long long a[3] ;
```

```
    typedef llong a[3] ;
```

```
    array a ;
```

```
    return 0;
```

```
}
```

타입의 재정의

- 2차원 배열의 재정의
 - 이중 재정의 방법과 같은 방법으로 해석

```
#include <stdio.h>

int main(void)
{
    typedef int array[3];

    typedef array matrix[2];

    matrix m;

    return 0;
}
```


타입의 재정의

해석 방법 2: 변수 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int array[3];
```

```
    typedef array matrix[2];
```

```
    matrix m;
```

```
    return 0;
```

```
}
```

배열의 경우
변수 대입법으로
해석해야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int array[3];
```

```
    typedef array m[2];
```

```
    matrix m;
```

```
    return 0;
```

```
}
```

타입의 재정의

해석 방법 2: 변수 대입법

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int array[3];
```

```
    typedef array m[2];
```

```
    matrix m;
```

```
    return 0;
```

```
}
```

배열의 경우
변수 대입법으로
해석해야 한다.

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef int m[2][3];
```

```
    typedef array m[2];
```

```
    matrix m;
```

```
    return 0;
```

```
}
```

타입의 재정의

포인터 타입 재선언

```
#include <stdio.h>
```

```
int main(void)
```

```
{
```

```
    typedef unsigned int UINT;
```

```
    UINT a;
```

```
    UINT* p;
```

```
    return 0;
```

```
}
```

*p를 UINT와 바꿔서 해석한다.

STUD 앞에 키워드 struct를 넣을 필요가 없음.

구조체 포인터 타입 재선언

```
#include <stdio.h>
```

```
struct stud {
```

```
    int id;
```

```
    char name;
```

```
    double grade;
```

```
};
```

```
int main(void)
```

```
{
```

```
    typedef struct stud STUD;
```

```
    STUD s;
```

```
    STUD* p;
```

```
    return 0;
```

```
}
```

구조체 선언도 선언이므로 외부영역에 기술가능

*p를 STUD와 바꿔서 해석한다.

복잡한 타입의 재정의

```
#include <stdio.h>

typedef int M[2][3];
void f(M m)
{
    printf("%d in f()\n", sizeof(M) );
    printf("%d in f()\n", sizeof(m) );
}

int main(void)
{
    M a = {
        {1,2,3},
        {4,5,6}
    };
    f(a);
    printf("%d in main()\n", sizeof(M) );
    printf("%d in main()\n", sizeof(a) );
    return 0;
}
```

$T(M) == \text{int}[3][2]$
 $\text{int}[3][2] \rightarrow \text{int}[3]*$
 $T(m) == \text{int}[3]*$
 $T(a) == \text{int}[3][2]$
 $\text{int}[3][2] \rightarrow \text{int}[3]*$

입출력 결과

```
24 in f()
4 in f()
24 in main()
24 in main()
계속하려면 아무 키나 누르십시오 . . .
```

```
#include <stdio.h>

typedef int F(int);
typedef F* PF;

int f1(int j) { return 1+j; }
int f2(int j) { return 9-j; }

void print(PF pf1, PF pf2, int n)
{
    int n1, n2, i, j;
    for(j=0; j<10; j=j+1) {
        n1=pf1(j);
        n2=pf2(j);
        for(i=0; i<n1; i=i+1) printf("* ");
        for(i=0; i<n2; i=i+1) printf(". ");
        printf("\n");
    }
}

int main(void)
{
    print(f1, f2, 10);
    return 0;
}
```

$T(F) == \text{int}(\text{int})$
 $T(PF) == \text{int}(\text{int})*$
 $T(pf1) == T(pf2) == \text{int}(\text{int})*$
 $T(f1) == T(f2) == \text{int}(\text{int})$
 $\text{int}(\text{int}) \rightarrow \text{int}(\text{int})*$

복잡한 타입의 재정의

```
#include <stdio.h>

typedef int F(int);
typedef F* PF;
typedef PF PFL[2];

int f1(int j) { return 1+j; }
int f2(int j) { return 9-j; }

void print(PFL pf_l, int n)
{
    int n1,n2,i,j;
    for(j=0;j<10;j=j+1) {
        n1=pf_l[0](j);
        n2=pf_l[1](j);
        for(i=0;i<n1;i=i+1) printf("* ");
        for(i=0;i<n2;i=i+1) printf(". ");
        printf("\n");
    }
}
```

$T(F) == \text{int}(\text{int})$
 $T(PF) == \text{int}(\text{int})^*$
 $T(PFL) == \text{int}(\text{int})^{*[2]}$

$\text{int}(\text{int})^{*[2]} \rightarrow \text{int}(\text{int})^{**}$
 $T(pf_l) == \text{int}(\text{int})^{**}$

```
int main(void)
{
    PFL pf_list = { &f1, &f2 };
    print(pf_list,10);
    return 0;
}
```

$T(pf_list) == \text{int}(\text{int})^{*[2]}$
 $T(\&f1) == T(\&f2) == \text{int}(\text{int})^*$

입출력 결과

```
* . . . . .
* * . . . . .
* * * . . . . .
* * * * . . . . .
* * * * * . . . . .
* * * * * * . . . . .
* * * * * * * . . . . .
* * * * * * * * . . . . .
* * * * * * * * * . . . . .
* * * * * * * * * * . . . . .
* * * * * * * * * * * . . . . .
* * * * * * * * * * * * . . . . .
* * * * * * * * * * * * * . . . . .
* * * * * * * * * * * * * * . . . . .
* * * * * * * * * * * * * * * . . . . .
계속하려면 아무 키나 누르십시오 . . .
```