



## -Part2-

# 제5장 포인터와 함수 그리고 void형 포인터

## 학습목차

5. 1 값에 의한 호출과 주소에 의한 호출

5. 2 주소를 반환하는 함수

5. 3 main() 함수에 인자가 있을 때

5. 4 void형 포인터란

## 들어가기에 앞서...

### ▶ 값에 의한 호출(Call By Value)

- ✓ 변수(메모리 공간에 저장된 값) 또는 값을 복사해서 함수 호출

### ▶ 주소에 의한 호출(Call By Reference)

- ✓ 주소(메모리 공간의 주소)를 참조해서 함수 호출

#### 함수의 출력, 입력 형태

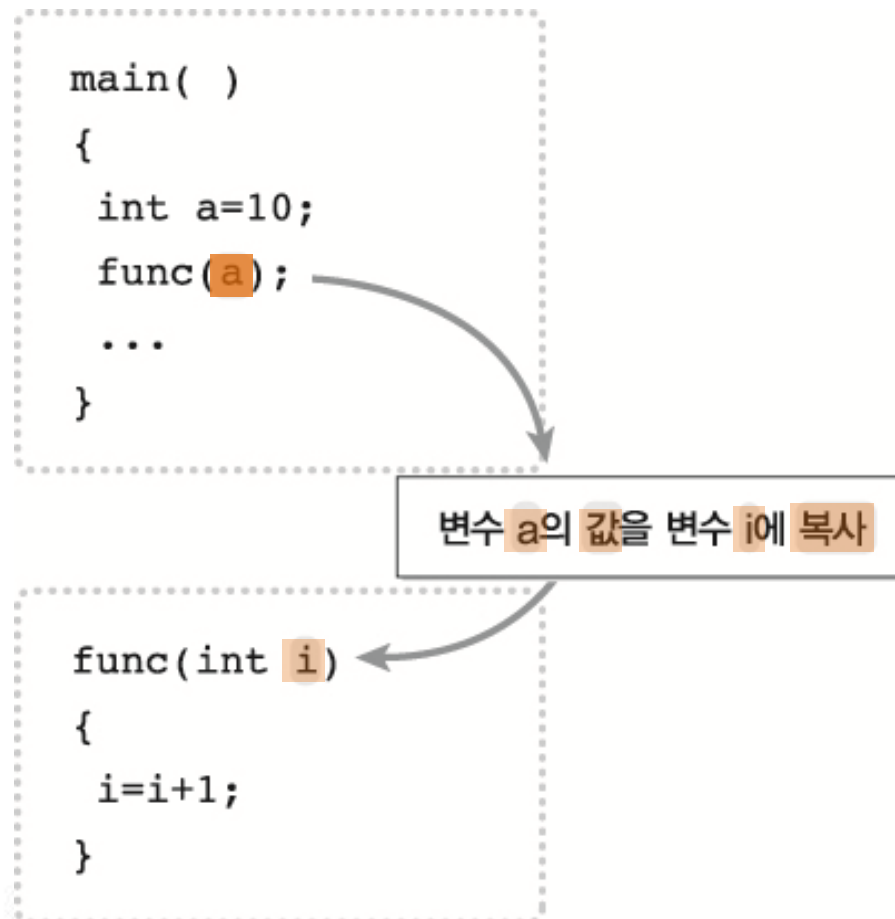
출력 형태 없음	입력 형태 없음	00 형태
출력 형태 없음	입력 형태 있음	01 형태
출력 형태 있음	입력 형태 없음	10 형태
출력 형태 있음	입력 형태 있음	11 형태

## 5.1 값에 의한 호출과 주소에 의한 호출

## 5.1 값에 의한 호출과 주소에 의한 호출 (1/10)

### ▶ 값에 의한 호출(Call by Value)

✓ 변수의 값을 복사해서 함수를 호출하는 방식



## 5.1 값에 의한 호출과 주소에 의한 호출 (2/10)---[5-1.c 실습]

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

```
int func(int i);    // 함수의 선언, 11 형태
```

```
void main( )
```

```
{
```

```
7행 int a=10;
```

```
8행 int result=0;
```

```
10행 result=func(a); // 값에 의한 함수 호출
```

```
printf("%d \n", result);
```

```
printf("%d \n", a);
```

```
}
```

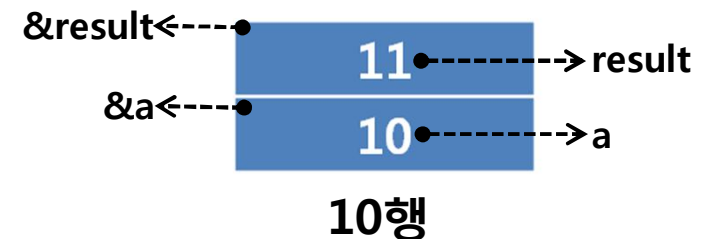
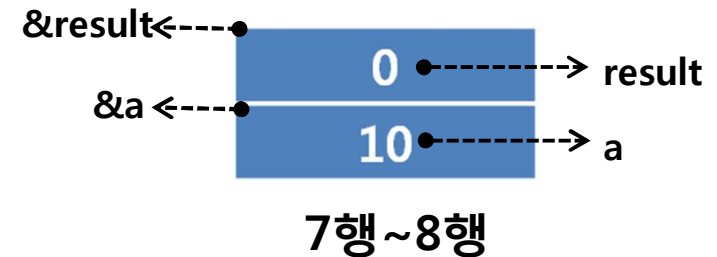
```
int func(int i)    // 함수의 정의
```

```
{
```

```
    i=i+1;
```

```
    return i;
```

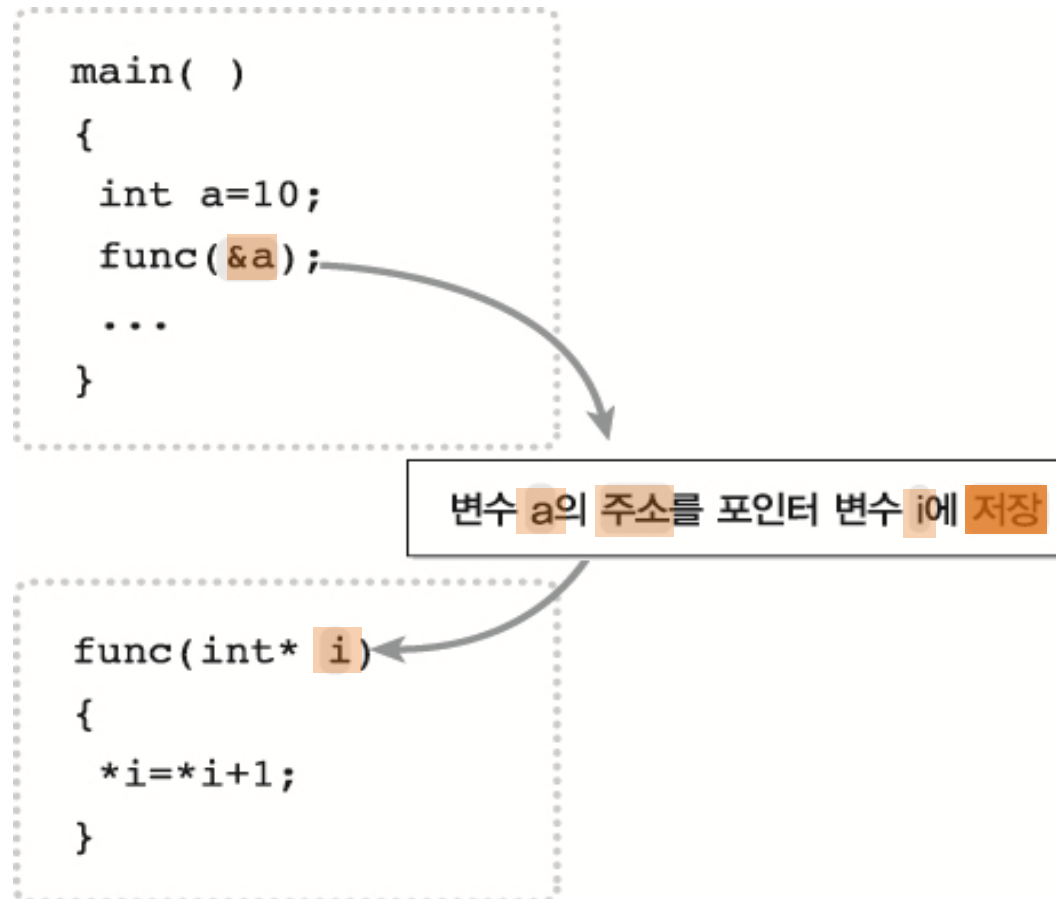
```
}
```



## 5.1 값에 의한 호출과 주소에 의한 호출 (3/10)

### ▶ 주소에 의한 호출(Call by Reference)

✓ 주소를 참조해서 함수를 호출하는 방식



## 5.1 값에 의한 호출과 주소에 의한 호출 (4/10)---[5-2.c 실습]

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

```
int func(int* i);           // 함수의 선언, 11 형태
```

```
void main( )
```

```
{
```

```
7행 int a=10;
```

```
8행 int result=0;
```

```
10행 result=func(&a);       // 주소에 의한 호출
```

```
printf("%d \n", result);
```

```
printf("%d \n", a);
```

```
}
```

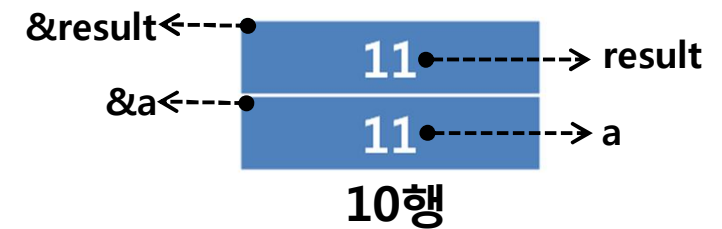
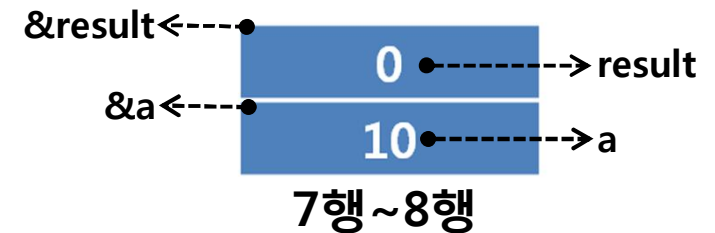
```
int func(int* i)           // 함수의 정의
```

```
{
```

```
    *i=*i+1;
```

```
    return *i;
```

```
}
```





## 5.1 값에 의한 호출과 주소에 의한 호출 (5/10)

### ▶ 값에 의한 호출(Call by value)의 문제

✓ '함수의 인자 전달에 사용되는 매개 변수가 많다.'

```
#include <stdio.h>

void func(int a1, int a2, int a3, int a4, int a5, int a6, int a7);

int main(void)
{
    int a=10, b=20, c=30, d=40, e=50, f=60, g=70;
    func(a, b, c, d, e, f, g);
    return 0;
}

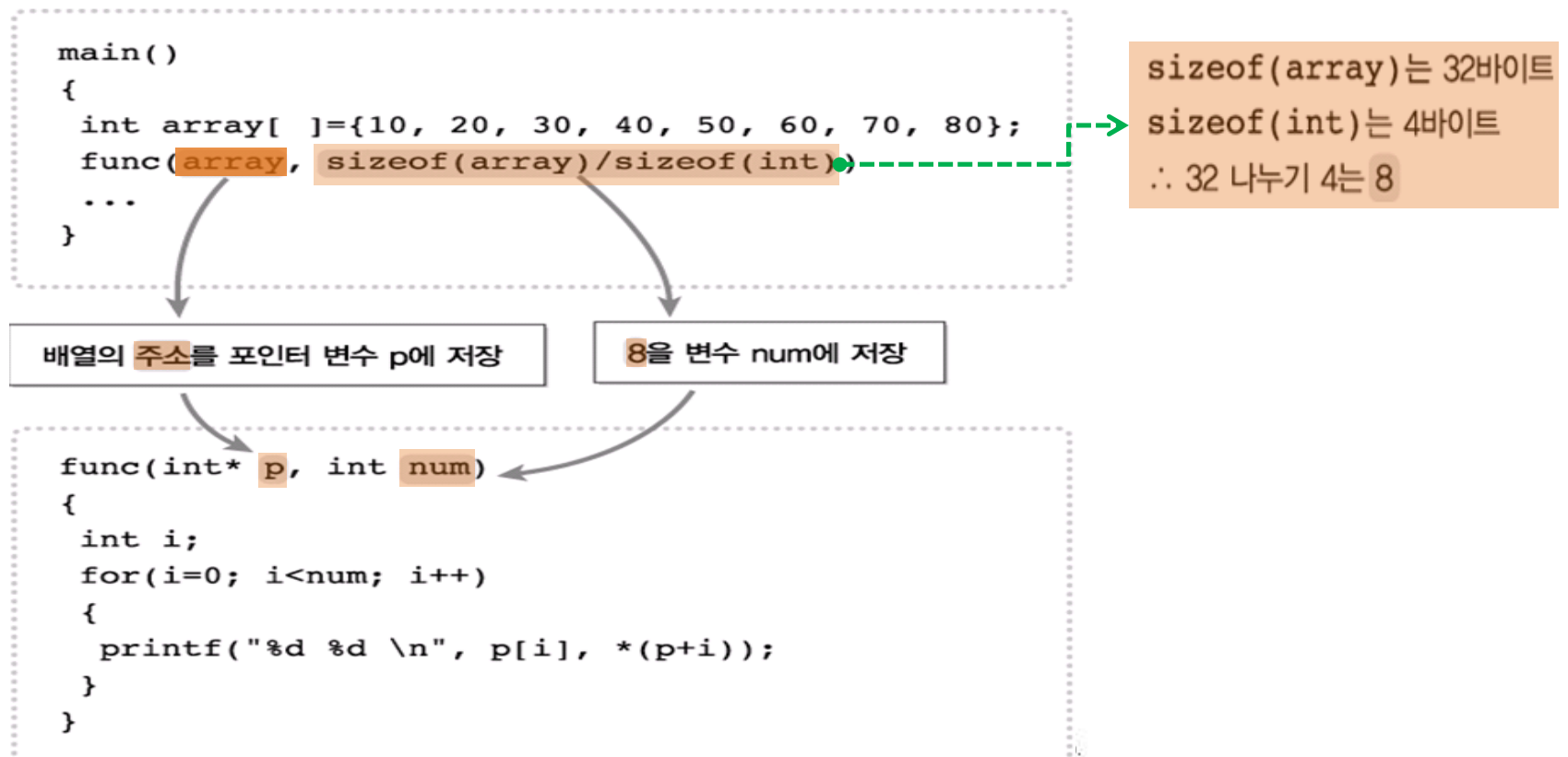
void func(int a1, int a2, int a3, int a4, int a5, int a6, int a7)
{
    printf("%d %d %d %d %d %d %d \n", a1, a2, a3, a4, a5, a6, a7);
}
```

## 5.1 값에 의한 호출과 주소에 의한 호출 (6/10)

### ▶ 주소에 의한 호출의 필요성

✓ '배열이나 구조체와 같은 데이터를 함수에 전달할 때 좋다.'

- 실행 시간 단축, 메모리 공간 적게 차지



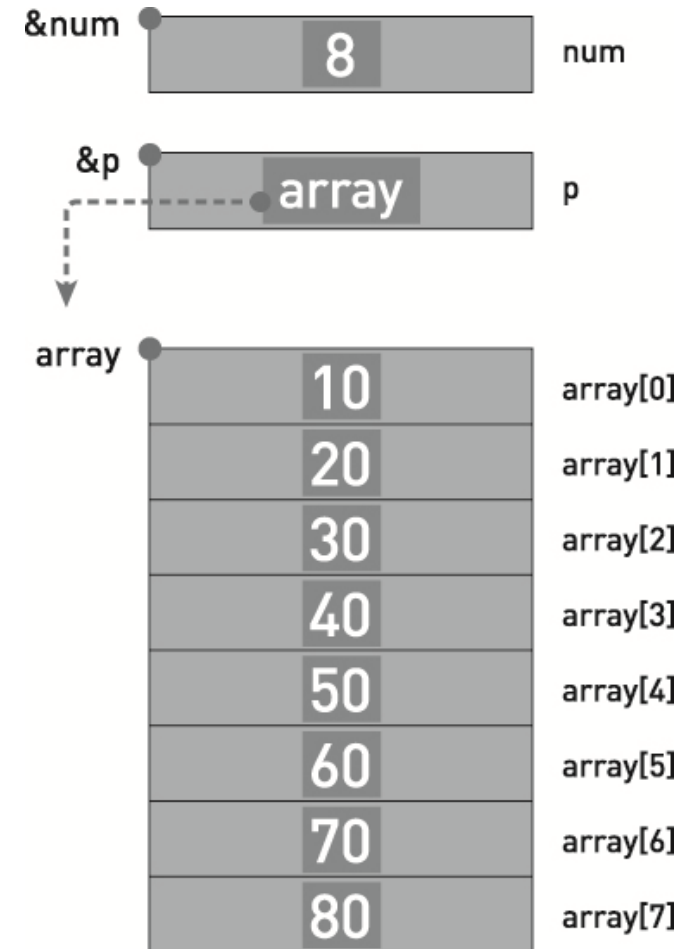
## 5.1 값에 의한 호출과 주소에 의한 호출 (7/10)---[5-3.c 실습]

```
#include <stdio.h>
void func(int* p, int num);           // 함수의 선언, 01 형태
int main(void)
{
    int array [ ]={10,20,30,40,50,60,70,80};
    func(array, sizeof(array)/sizeof(int)); // 함수의 호출
    return 0;
}

12행 void func(int* p, int num)       // 함수의 정의
{
    int i;
    for(i=0; i<num; i++)
    {
        printf("%d %d \n", p[i], *(p+i)); // p[i] == *(p+i)
    }
}
```

## 5.1 값에 의한 호출과 주소에 의한 호출 (8/10)---[5-3.c 분석]

```
void func(int* p, int num)
{
    int i;
    for(i=0; i<num; i++)
    {
        printf("%d %d \n", p[i], *(p+i) );
    }
}
```



12행

## 5.1 값에 의한 호출과 주소에 의한 호출 (9/10)---[5-4.c 실습]

### ▶ 배열 포인터를 이용한 주소에 의한 호출

```
...  
void func(int(*p)[4], int num1, int num2);           // 함수의 선언, 배열 포인터 변수  
int main(void)  
{  
    int array[2][4]={10,20,30,40,50,60,70,80};  
    func(array, sizeof(array)/16, sizeof(array)/8); // 함수의 호출  
    return 0;  
}  
  
void func(int(*p)[4], int num1, int num2)           // 함수의 정의  
{  
    int i, j;  
    for(i=0; i<num1; i++)  
    {  
        for(j=0; j<num2; j++)  
        {  
            printf("%d ", p[i][j]);  
        }  
        printf("\n");  
    }  
}
```

## 5.1 값에 의한 호출과 주소에 의한 호출 (10/10)---[5-5.c 실습]

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

```
void func(int* p);           // 함수의 선언
```

```
int main(void)
```

```
{  
    int array[2][4]={10,20,30,40,50,60,70,80};  
    func(array);             // 함수의 호출  
    return 0;  
}
```

```
void func(int* p)           // 함수의 정의  
{  
    printf("%d %d %d %d %d %d %d %d \n", p[0],p[1],p[2],p[3],p[4],p[5],p[6],p[7]);  
    printf("%d %d %d %d \n", p[0][0], p[0][1], p[0][2], p[0][3]); // 에러  
    printf("%d %d %d %d \n", p[1][0], p[1][1], p[1][2], p[1][3]); // 에러  
}
```

error C2109: 첨자는 배열 또는 포인터 형식을 사용해야 합니다.

## 5.2 주소를 반환하는 함수

## 5.2 주소를 반환하는 함수 (1/8)

### ▶ 주소 반환(**return**)의 필요성과 주의 사항

- ✓ 필요성: 대량의 데이터를 반환(**return**)할 때 사용
- ✓ 주의 사항: 지역 변수의 주소를 반환(**return**)하면 경고 발생
  - 경고 문제 해결 방법: **static** 변수의 사용



## 5.2 주소를 반환하는 함수 (2/8)---[5-6.c 실습]

```
#include <stdio.h>
int* input( );    // 함수의 선언
int main(void)
{
    int* p=NULL;

    p=input( );    // 함수의 호출
    printf("%d \n", *p);
    return 0;
}

int* input( )    // 함수의 정의
{
    int num1;
    scanf("%d", &num1);
    return &num1;
}
```

num1은 지역변수(경고 발생)

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

## 5.2 주소를 반환하는 함수 (3/8)

### ▶ 주소 반환(return) 시 유용한 정적(static)변수

- ✓ 정적(static) 변수: 함수가 종료된 후에도 메모리 공간이 소멸되지 않음
- ✓ 지역 변수의 주소를 반환해서 생기는 경고 문제 해결

## 5.2 주소를 반환하는 함수 (4/8)---[5-7.c 실습]

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

```
int* input( );           // 함수의 선언
```

```
int main(void)
{
    int* p=NULL;
```

```
    p=input( );           // 함수의 호출
    printf("%d \n", *p);
    return 0;
}
```

```
int* input( )           // 함수의 정의
{
```

```
    static int num1;     // 정적 변수 선언
```

```
    scanf("%d", &num1);
    return &num1;
```

```
}
```

num1은 정적 변수(경고 제거)

## 5.2 주소를 반환하는 함수 (5/8)---[5-8.c 실습]

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

```
int* func( );    // 함수의 선언
```

```
int main(void)
```

```
{
```

```
    int* p=NULL;
```

```
    → p=func( );    // 함수의 호출
```

```
    printf("%d %d %d %d \n", p[0], p[1], p[2], p[3]);
```

```
    printf("%d %d %d %d \n", *(p+0), *(p+1), *(p+2), *(p+3));
```

```
    return 0;
```

```
}
```

```
int* func( )    // 함수의 정의
```

```
{
```

```
    int array[ ]={10, 20, 30, 40};
```

```
    return array;    // 경고 발생
```

```
}
```

## 5.2 주소를 반환하는 함수 (6/8)---[5-9.c 실습]

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

```
int* func( );    // 함수의 선언
```

```
int main(void)
```

```
{
```

```
    int* p=NULL;
```

```
    -> p=func( );    // 함수의 호출
```

```
    printf("%d %d %d %d \n", p[0], p[1], p[2], p[3]);
```

```
    printf("%d %d %d %d \n", *(p+0), *(p+1), *(p+2), *(p+3));
```

```
    return 0;
```

```
}
```

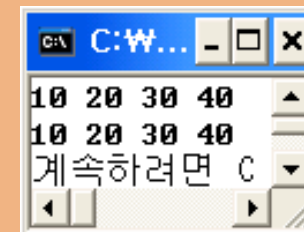
```
int* func( )    // 함수의 정의
```

```
{
```

```
    static int array[ ]={10, 20, 30, 40};
```

```
    return array;    // 경고 제거
```

```
}
```



## 5.2 주소를 반환하는 함수 (7/8)---[5-10.c 실습]

```
#include <stdio.h>

char* string1(void); // 함수의 선언
char* string2(void); // 함수의 선언

int main(void)
{
    char* p1=NULL;
    char* p2=NULL;

    p1=string1( ); // 함수의 호출
    p2=string2( ); // 함수의 호출

    printf("%s \n", p1);
    printf("%s \n", p2);
    return 0;
}
```

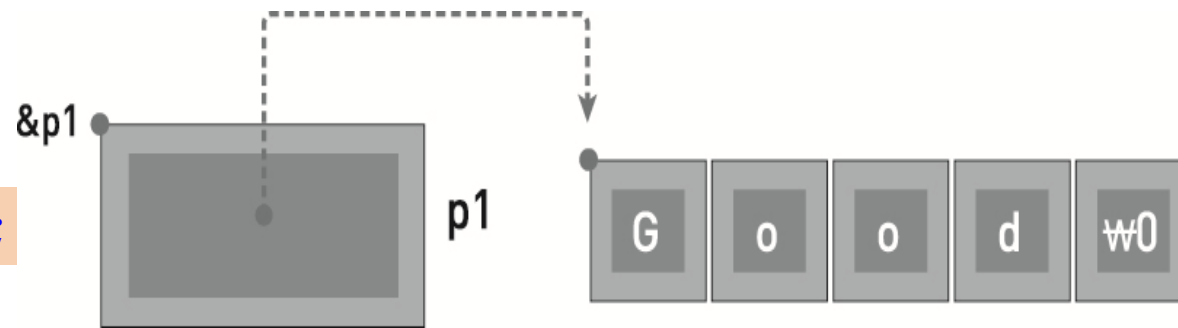
```
char* string1(void) // 함수의 정의
{
    static char str[ ]="Good";
    return str; // 시작 주소 반환
}
```

```
char* string2(void) // 함수의 정의
{
    static char str[ ]="morning";
    return str; // 시작 주소 반환
}
```

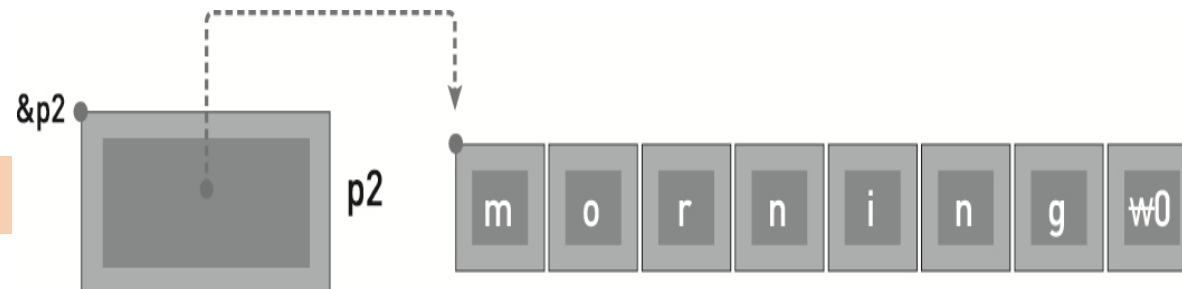
```
C:\WINDOWS\system32...
Good
morning
계속하려면 아무 키나 누르십시오
```

## 5.2 주소를 반환하는 함수 (8/8)---[5-10.c 분석]

```
p1=string1( );
```



```
p2=string2( );
```



## 5.3 main() 함수에 인자가 있을 때

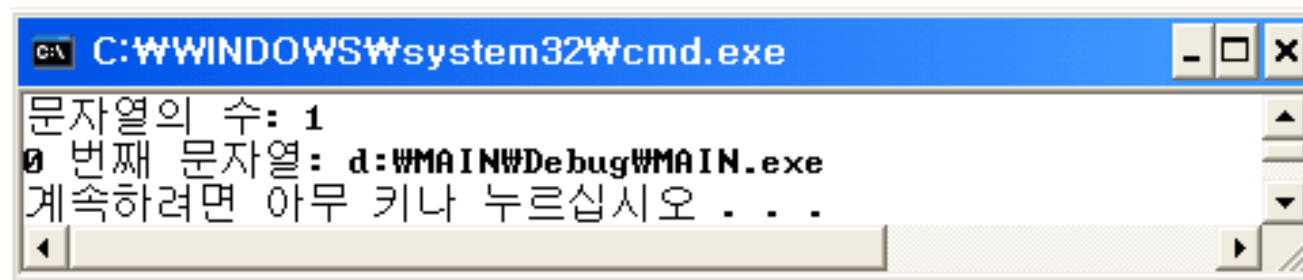


## 5.3 main() 함수에 인자가 있을 때 (1/7)---[5-11.c 실습]

```
#include <stdio.h>

int main(int argc, char* argv[ ])
{
    int i=0;
    printf("문자열의 수 : %d \\\n", argc);

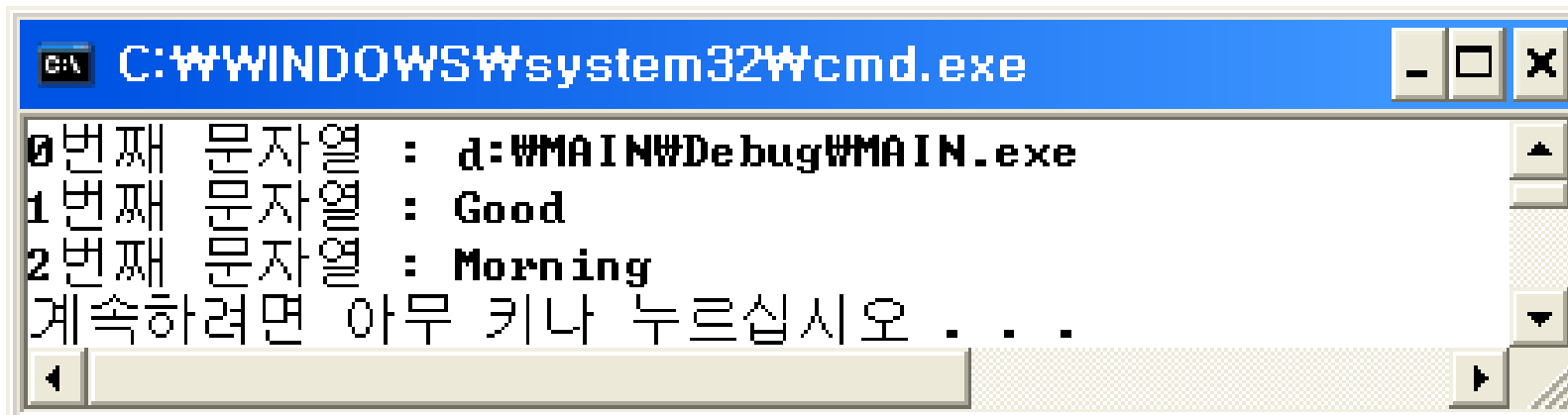
    for(i=0; i<argc; i++)
    {
        printf("%d번째 문자열 : %s \\\n", i, argv[i]);
    }
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
문자열의 수: 1
0 번째 문자열: d:\MAIN\Debug\MAIN.exe
계속하려면 아무 키나 누르십시오 . . .
```

- 26 -

## 5.3 main() 함수에 인자가 있을 때 (3/7)---[5-11.c 실습]



```
C:\WINDOWS\system32\cmd.exe
0번째 문자열 : d:\MAIN\Debug\MAIN.exe
1번째 문자열 : Good
2번째 문자열 : Morning
계속하려면 아무 키나 누르십시오 . . .
```

## 5.3 main() 함수에 인자가 있을 때 (4/7)---[5-12.c 실습]

```
#include <stdio.h>

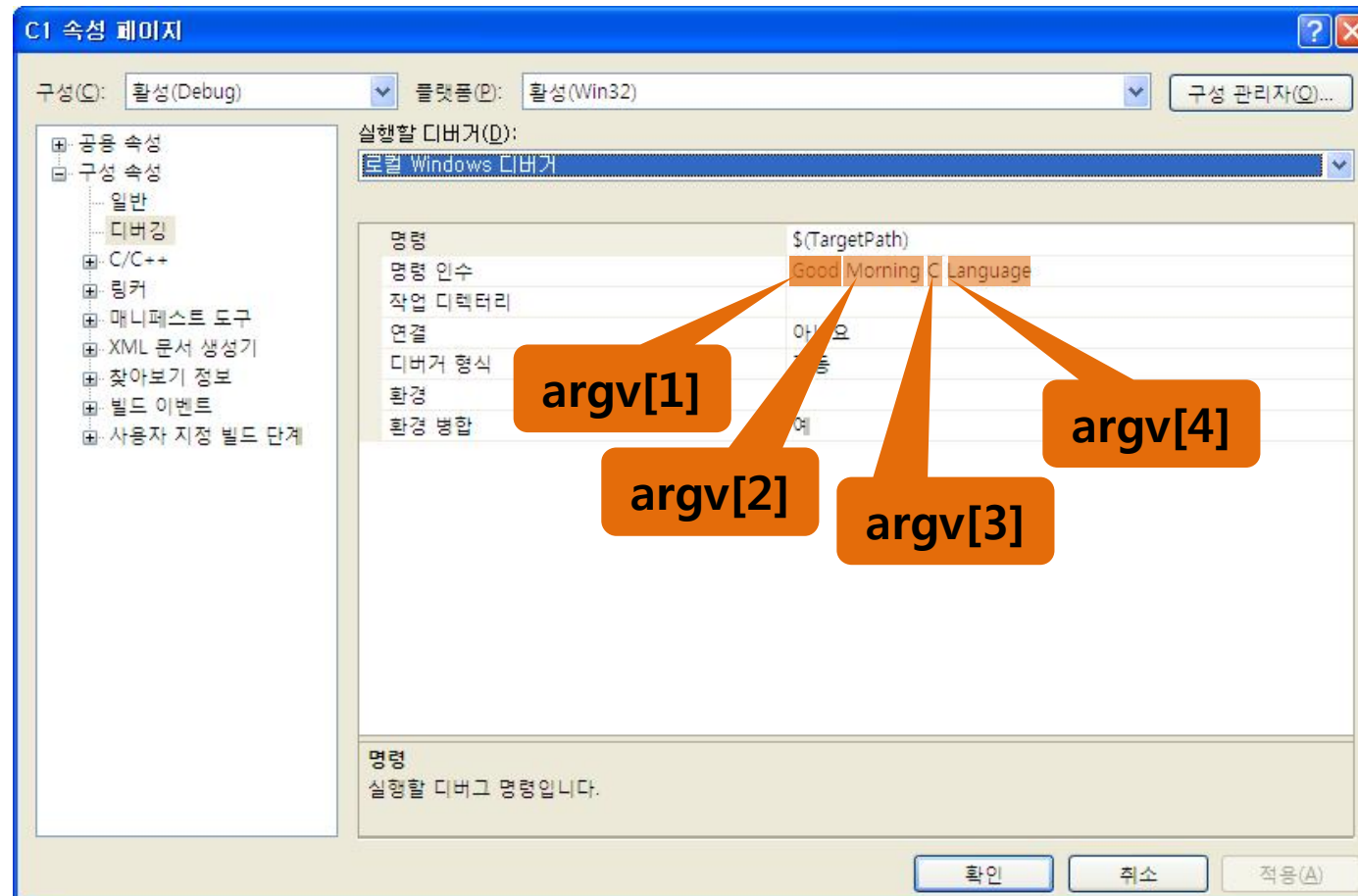
int main(int argc, char* argv[ ])
{
    int i=0;
    printf("문자열의 수 : %d \n", argc);

    for(i=0; i<argc; i++)
    {
        printf("argv[%d] : %s \n", i, argv[i]);
    }
    return 0;
}
```

## 5.3 main() 함수에 인자가 있을 때 (5/7)---[5-12.c 실습]

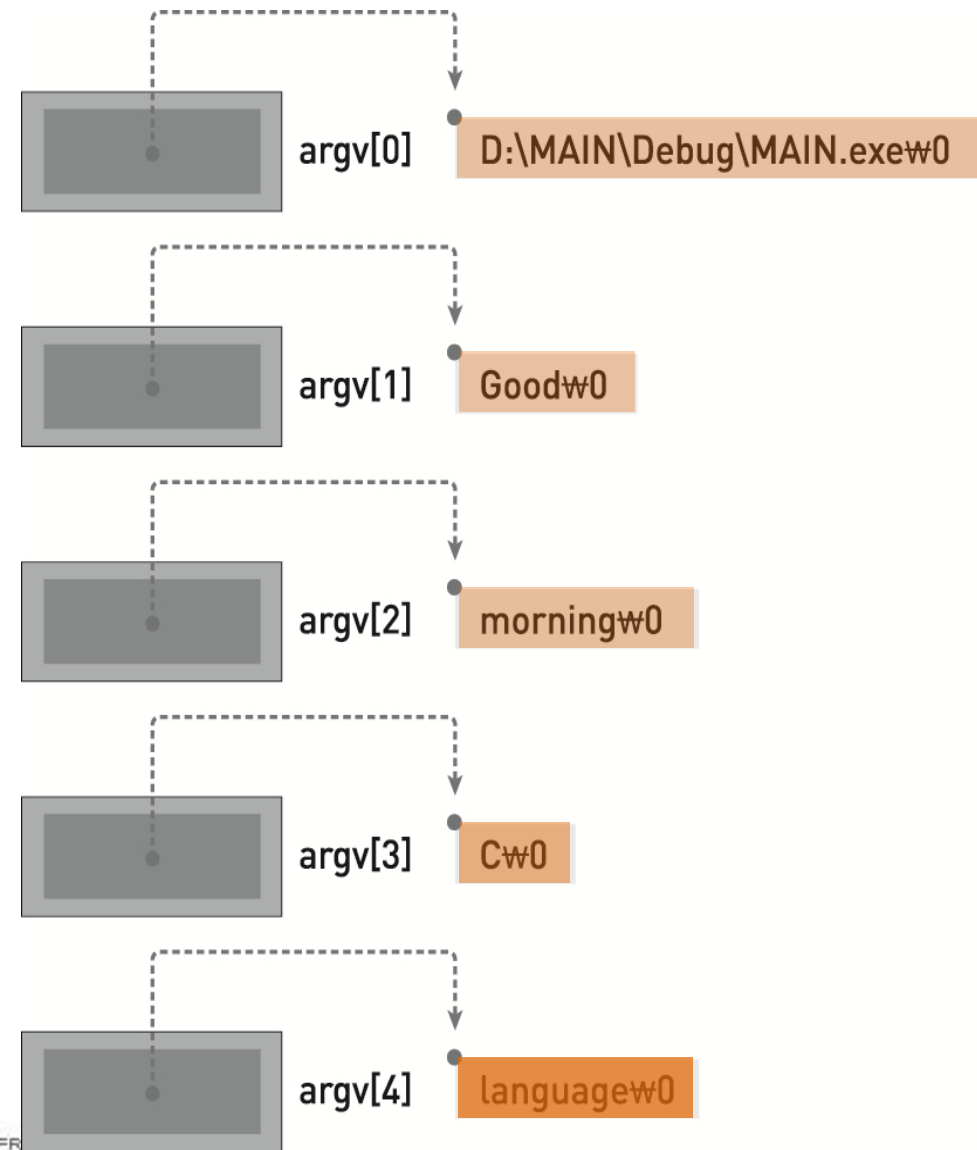
### ▶ Visual C++ 개발 환경을 이용한 main()함수의 인자 전달

- ✓ **방법1** : [Alt + F7] → 구성 속성 → 디버깅
- ✓ **방법2**: 메뉴 → [프로젝트] → 속성 → 구성 속성 → 디버깅



## 5.3 main() 함수에 인자가 있을 때 (6/7)---[5-12.c 분석]

```
C:\ C:\WINDOWS\system32\cmd.  
문자열의 수 : 5  
argv[0] : d:\MAIN\Debug\MAIN.exe  
argv[1] : Good  
argv[2] : Morning  
argv[3] : C  
argv[4] : Language  
계속하려면 아무 키나 누르십시오
```



## 5.3 main() 함수에 인자가 있을 때 (7/7)---[5-13.c 실습]

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

```
int main(int argc, char* argv[ ])
{
    int i=0;
    if(argc>4)
    {
        printf("문자열의 수가 너무 많습니다. \n");
        printf("프로그램을 종료합니다. \n");
        return 1;
    }

    printf("0번째 문자열 : %s \n", argv[0]);
    printf("1번째 문자열 : %s \n", argv[1]);
    printf("2번째 문자열 : %s \n", argv[2]);
    printf("3번째 문자열 : %s \n", argv[3]);

    return 0;
}
```

명령 프롬프트를 이용한  
main()함수 인자 전달

```
C:\WINDOWS\system32\cmd.exe
C 드라이브의 볼륨에는 이름이 없습니다.
볼륨 일련 번호: F85B-7529

C:\MAINWDebug 디렉터리

2011-02-19 오후 12:18 <DIR> .
2011-02-19 오후 12:18 <DIR> ..
2011-02-19 오후 12:18      30,720 MAIN.exe
2011-02-19 오후 12:18     304,248 MAIN.ilk
2011-02-19 오후 12:18     388,096 MAIN.pdb
                3개 파일              723,064 바이트
                2개 디렉터리 97,579,696,128 바이트 남음

C:\MAINWDebug>MAIN.exe Good morning
0번째 문자열 : MAIN.exe
1번째 문자열 : Good
2번째 문자열 : morning
3번째 문자열 : <null>
```

## 5.4 void형 포인터란



## 5.4 void형 포인터란 (1/5)

### ▶ void형 포인터

- ✓ 자료형을 지정하지 않은 포인터 변수
- ✓ 어떤 자료형의 주소라도 저장할 수 있는 포인터 변수
- ✓ 주의사항: \* 연산자로 값을 접근하려면 강제 형변환 필요

```
char* p = NULL;
```

```
int* p = NULL;
```

```
double* p = NULL;
```

```
void* p = NULL;
```

void형 포인터



## 5.4 void형 포인터란 (2/5)---[5-14.c 실습]

```
#include <stdio.h>
int main(void)
{
    char c=3;
    double d=3.1;

    void* vx;

    vx=&c;        // char형 변수 c의 주소를 저장
    printf("vx의 주소 값 : %x \n", vx);
    // printf("vx의 값 : %d \n", *vx); // 에러

    vx=&d;        // double형 변수 d의 주소를 저장
    printf("vx의 주소 값 : %x \n", vx);
    // printf("vx의 값 : %lf \n", *vx); // 에러
    return 0;
}
```

## 5.4 void형 포인터란 (3/5)---[5-15.c 실습]

```
#include <stdio.h>
void main( )
{
    char c=3;
    double d=3.1;

    void* vx=NULL;

    vx=&c;
    printf("vx가 저장한 값 : %x \\\n", vx);
    printf("*vx의 값 : %d \\\n", *(char*)vx);    // 강제 형변환

    vx=&d;
    printf("vx가 저장한 값 : %x \\\n", vx);
    printf("*vx의 값 : %lf \\\n", *(double*)vx); // 강제 형변환
}
```

## 5.4 void형 포인터란 (4/5)---[5-15.c 분석]

**vx=&c;**

printf("vx가 저장한 값 : %x \\\n", vx);

printf("\*vx의 값 : %d \\\n", **\*(char\*)vx**);

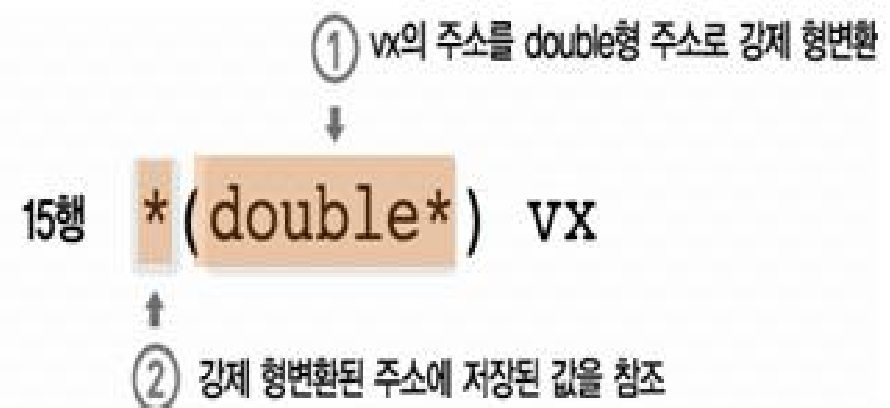
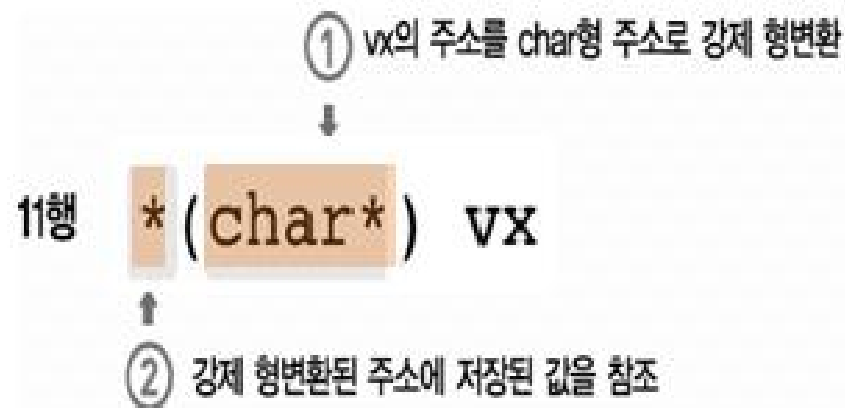
// 강제 형변환

**vx=&d;**

printf("vx가 저장한 값 : %x \\\n", vx);

printf("\*vx의 값 : %lf \\\n", **\*(double\*)vx**);

// 강제 형변환



## 5.4 void형 포인터란 (5/5)---[5-16.c 실습]

```
...
char c=3;
double d=3.1;
void* vx=NULL;

vx=&c;
printf("vx가 저장한 주소 : %x \n", vx);
printf("*vx의 값 : %d \n", *(char*)vx); // 강제 형변환(char*)

vx=&d;
printf("vx가 저장한 주소 : %x \n", vx);
printf("*vx의 값 : %lf \n", *(double*)vx); // 강제 형변환(double*)

vx=&c;
*(char*)vx=5; // 강제 형변환(char*)
printf("c가 저장한 값 : %d \n", c);
printf("*vx의 값 : %d \n", *(char*)vx); // 강제 형변환(char*)

vx=&d;
*(double*)vx=5.1; // 강제 형변환(double*)
printf("d가 저장한 값 : %lf \n", d);
printf("*vx의 값 : %lf \n", *(double*)vx); // 강제 형변환(double*)
```

## 공부한 내용 떠올리기

- ▶ 값에 의한 호출과 주소에 의한 호출에 대한 특징과 차이점
- ▶ 주소를 반환하는 함수
- ▶ 주소를 반환할 때 정적 변수의 유용성
- ▶ 문자열 배열의 시작 주소를 반환하는 함수
- ▶ `main( )` 함수의 인자 전달과 역할
- ▶ `void`형 포인터
- ▶ `void`형 포인터의 강제 형변환

## 홀라후프 (출처: 사랑과 지혜의 탈무드)

