기초프로그래밍

제14장 포인터와 함수 그리고 void형 포인터

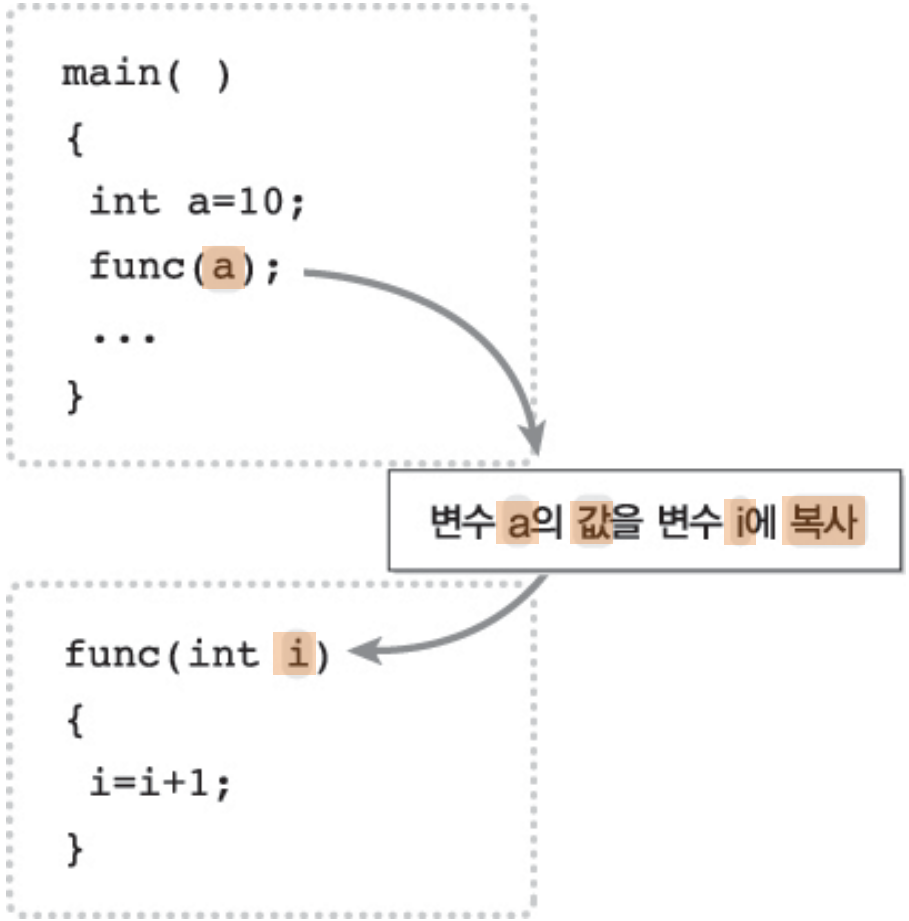
Sangsoo Lim
CSAI
Dongguk University

차례

- 값에 의한 호출과 주소에 의한 호출
- 주소를 반환하는 함수
- main() 함수의 인자
- void형 포인터

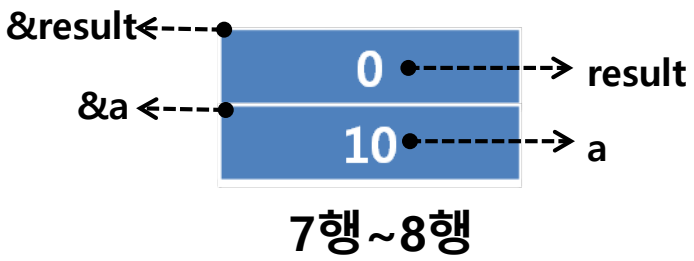
값에 의한 호출과 주소에 의한 호출

- **값에 의한 호출(Call by Value)**
  - 변수의 **값을 복사해서 함수를 호출**하는 방식

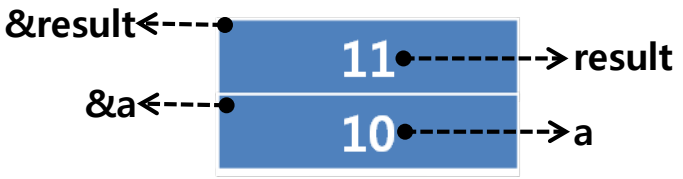


값에 의한 호출과 주소에 의한 호출

```c
/* 14-1.c */
#include <stdio.h>

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

void main( )
{
   int a=10;
   int result=0;

   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;
}
```



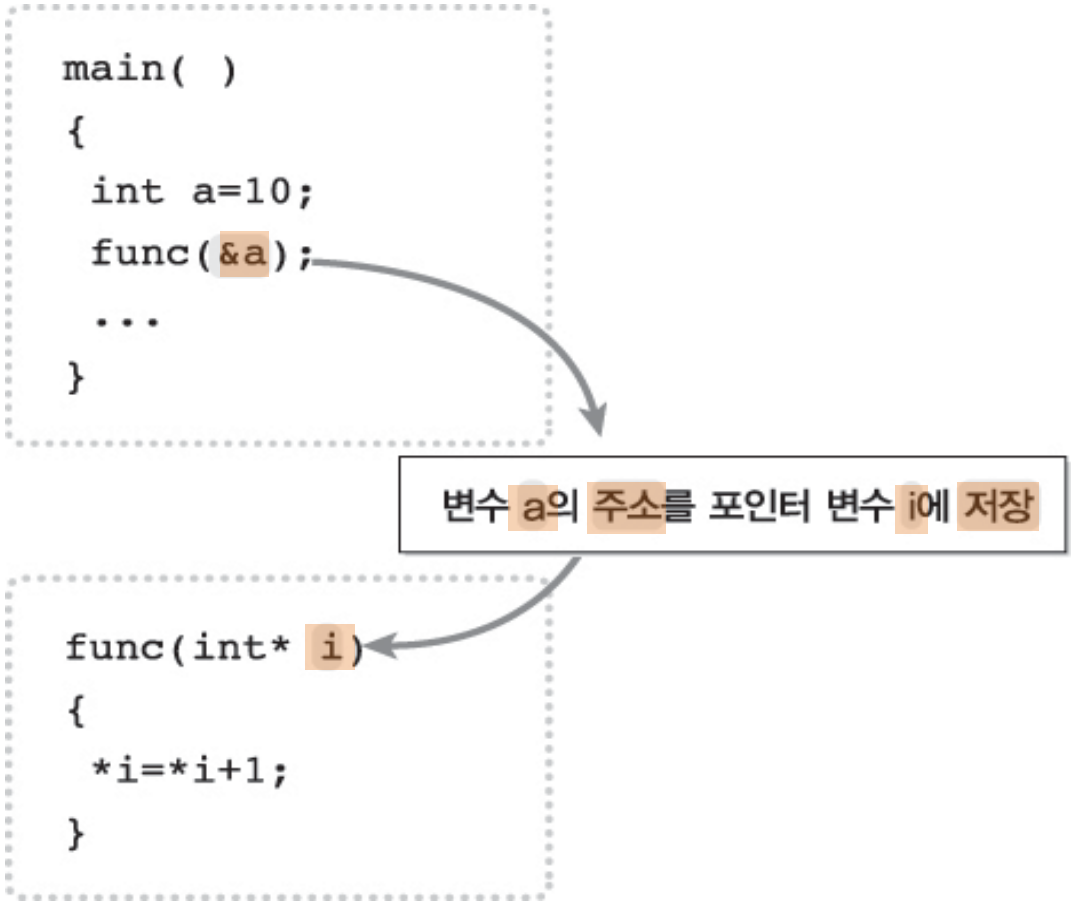
7행~8행



10행

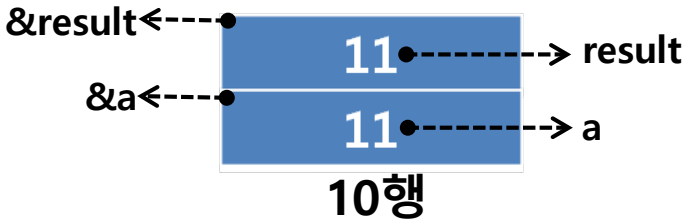
값에 의한 호출과 주소에 의한 호출

- **주소에 의한 호출(Call by Reference)**
  - **주소를 참조**해서 함수를 호출하는 방식



값에 의한 호출과 주소에 의한 호출

```c
/* 14-2.c */
#include <stdio.h>

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

void main( )
{
   int a=10;
   int result=0;

   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;
}
```



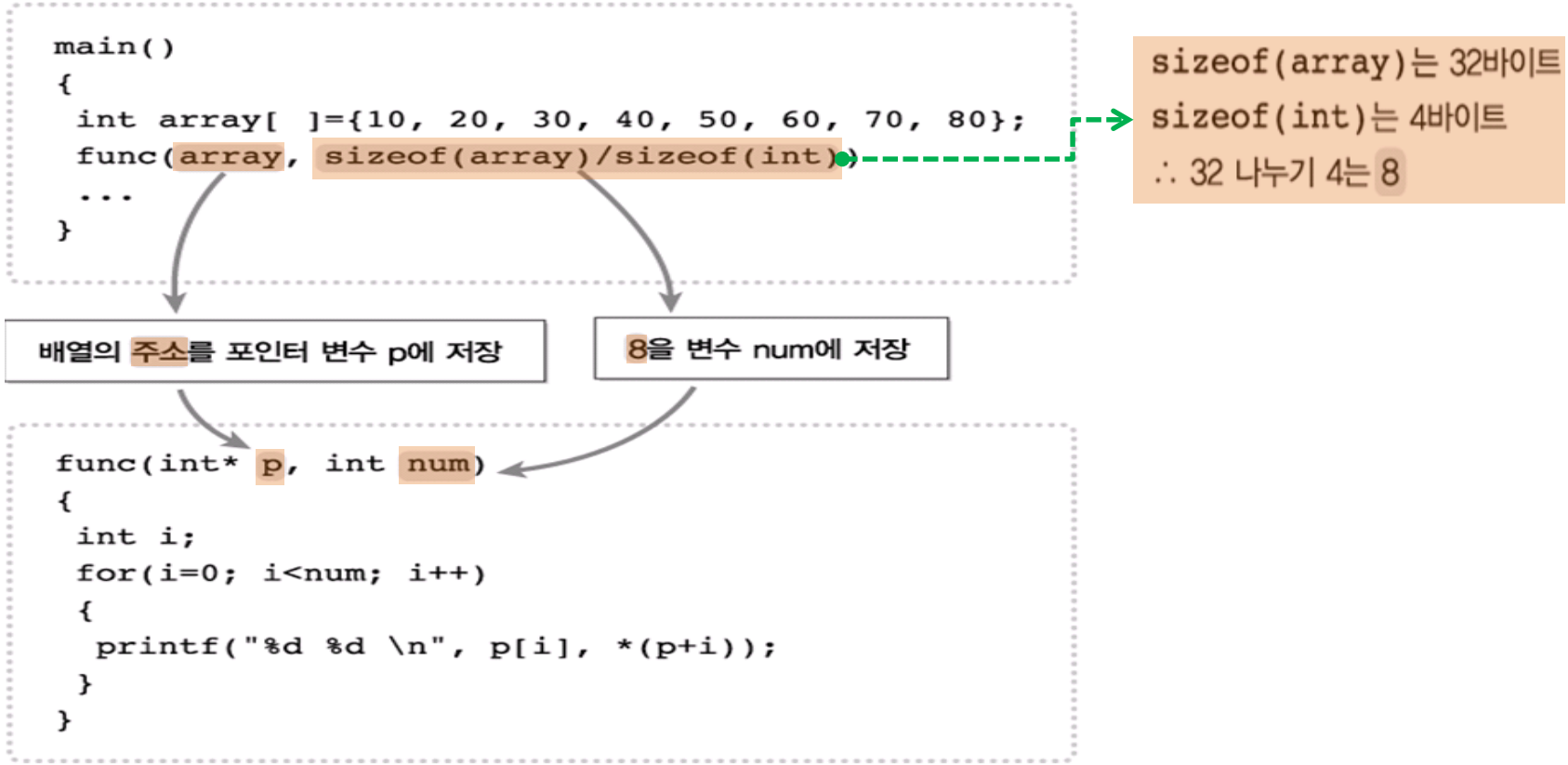
값에 의한 호출과 주소에 의한 호출

- **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);
}
```

값에 의한 호출과 주소에 의한 호출

- **주소에 의한 호출의 필요성**
  - 배열이나 구조체와 같은 데이터를 함수에 전달할 때 좋다.
    - 실행 시간 단축, 메모리 공간 적게 차지

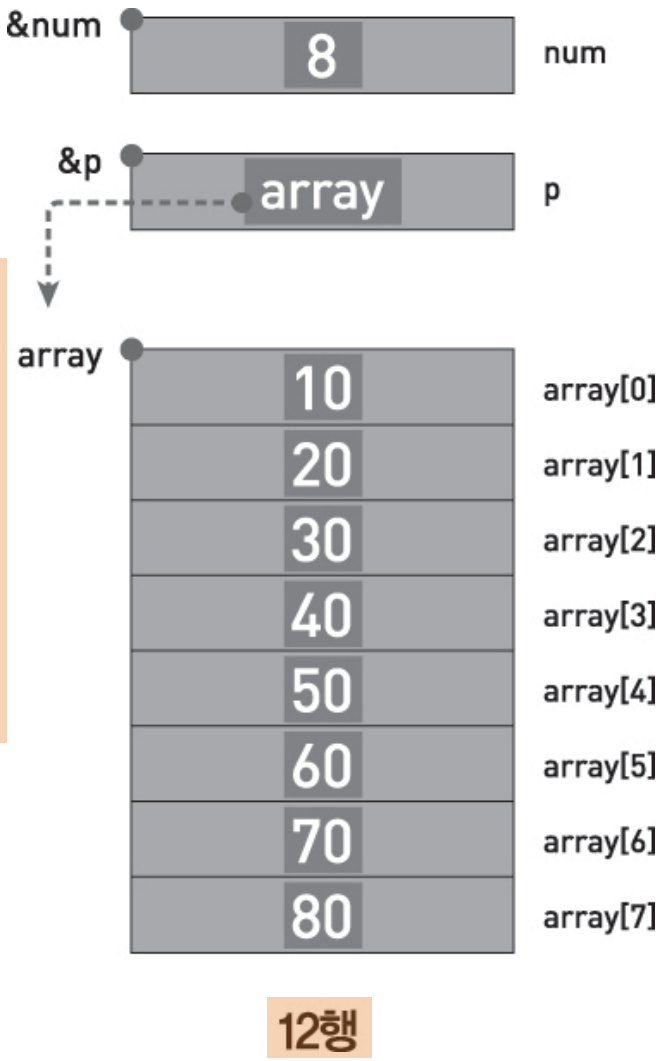


값에 의한 호출과 주소에 의한 호출

```c
/* 14-3.c */
#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;
}

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)
   }
}
```

값에 의한 호출과 주소에 의한 호출

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



12행

값에 의한 호출과 주소에 의한 호출

- **배열 포인터를 이용한 주소에 의한 호출**

```c
#include <stdio.h>

// Function declaration with array pointer variable
void func(int (*p)[4], int num1, int num2);

int main(void)
{
   int array[2][4] = {10, 20, 30, 40, 50, 60, 70, 80};

   // Correctly calculate the number of rows and columns
   int num1 = sizeof(array) / sizeof(array[0]);
   int num2 = sizeof(array[0]) / sizeof(array[0][0]);

   // Function call
   func(array, num1, num2);

   return 0;
}
```

```c
// Function definition
void func(int (*p)[4], int num1, int num2)
{
   int i, j;

   for (i = 0; i < num1; i++)
   {
      for (j = 0; j < num2; j++)
      {
         printf("%d ", p[i][j]); // Access elements in the 2D array
      }
      printf("\n");
   }
}
```

값에 의한 호출과 주소에 의한 호출

```c
/* 14-5.c */
#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
}
```

값에 의한 호출과 주소에 의한 호출 (수정코드)

```c
#include <stdio.h>

// Function declaration
void func(int (*p)[4]);  // Function now takes a pointer to a 1D array of 4 integers

int main(void)
{
   int array[2][4] = {10, 20, 30, 40, 50, 60, 70, 80};
   func(array);  // Function call passing the 2D array
   return 0;
}

// Function definition
void func(int (*p)[4])
{
   // Access elements row-wise using proper indexing
   printf("%d %d %d %d %d %d %d %d \n",
          p[0][0], p[0][1], p[0][2], p[0][3],
          p[1][0], p[1][1], p[1][2], p[1][3]);

   // Access rows separately
   printf("%d %d %d %d \n", p[0][0], p[0][1], p[0][2], p[0][3]);  // First row
   printf("%d %d %d %d \n", p[1][0], p[1][1], p[1][2], p[1][3]);  // Second row
}
```

주소를 반환하는 함수

- 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

주소를 반환하는 함수

```c
/* 14-6.c */
#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)

주소를 반환하는 함수

- **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**

주소를 반환하는 함수

```c
/* 14-7.c */
#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)

주소를 반환하는 함수

```c
/* 14-8.c */
#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
}
```

주소를 반환하는 함수

```c
/* 14-9.c */
#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
}
```

```
10 20 30 40
10 20 30 40
```

주소를 반환하는 함수

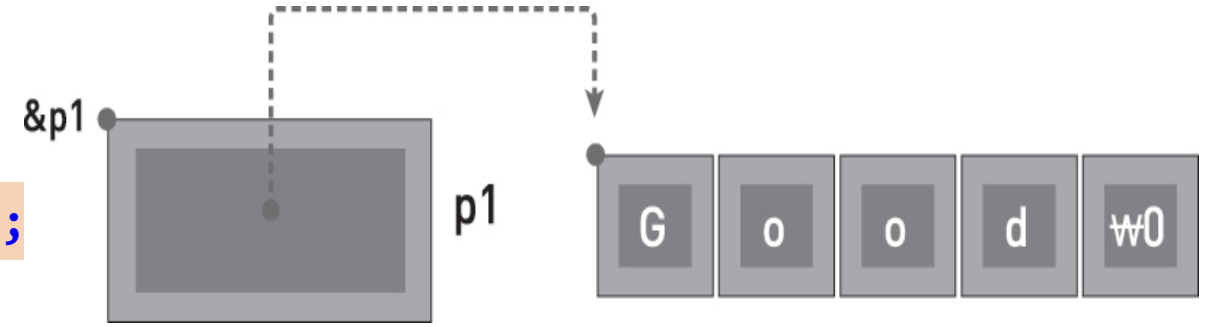
```c
/* 14-10.c */
#include <stdio.h>

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;
}
```

```c
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
}
```

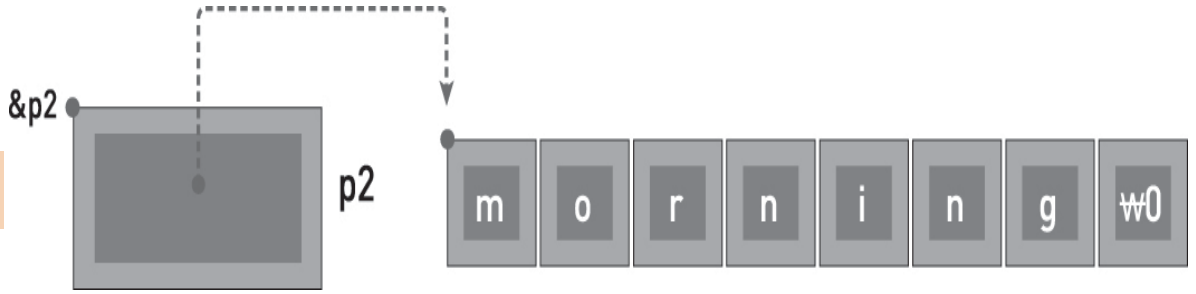
```
Good
morning
```

주소를 반환하는 함수

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




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



main() 함수에 인자가 있을 때

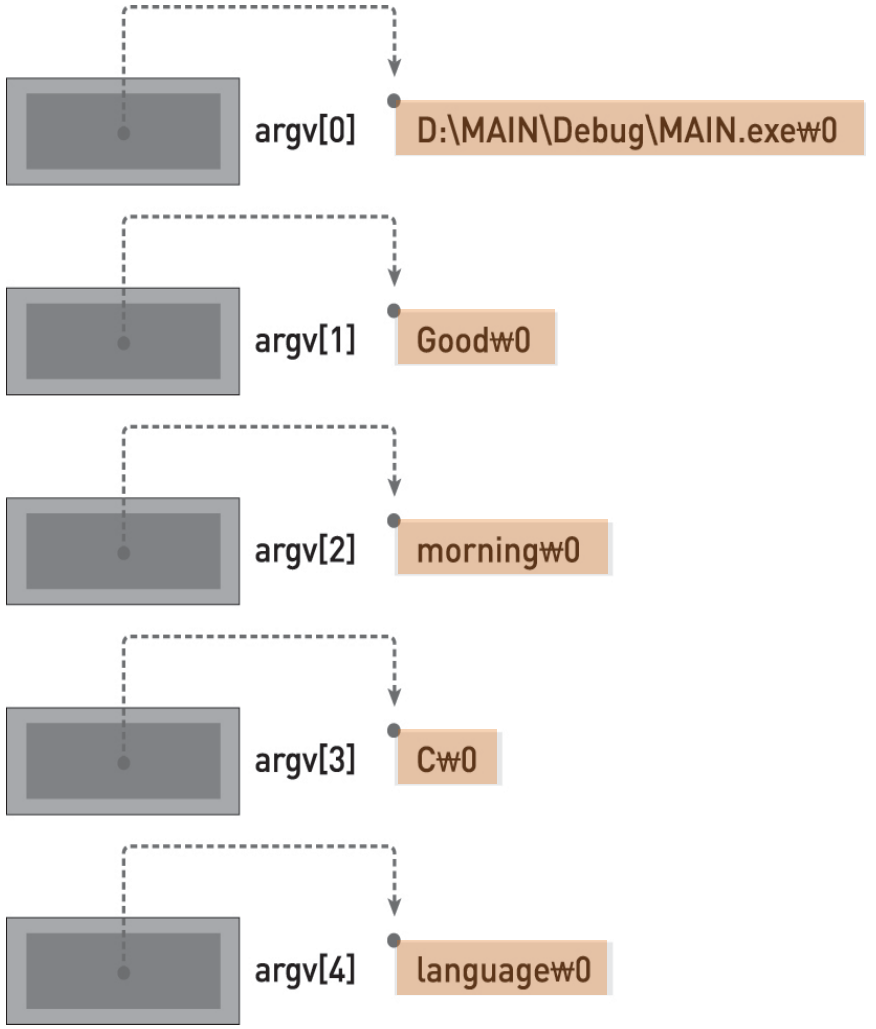
```c
/* 14-11.c */
#include <stdio.h>

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;
}
```

```
./14-11 Good morning C language
```

main() 함수에 인자가 있을 때

```c
/* 14-12.c */
#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;
}
```

./14-12 Good morning C language



main() 함수에 인자가 있을 때

```c
/* 14-13.c */
#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;
}
```

void형 포인터란

- **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 포인터란

```c
/* 14-14.c */
#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;
}
```

void형 포인터란

```c
/* 14-15.c */
#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
}
```

```
vx가 저장한 값 : af244407
*vx의 값 : 3
vx가 저장한 값 : af244408
*vx의 값 : 3.100000
```

void형 포인터란

```c
/* 14-16.c */
   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*)
```

Summary

- 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
- void형 포인터
- void형 포인터의 강제 형변환