COMP319 Algorithms
Lecture 1

C programming review, part 1

Instructor: Gil-Jin Jang **장길진**

Original slides:

한빛아카데미(주)

C로 배우는 쉬운 자료구조, 개정 3판, 이지영

Hanbit Academy

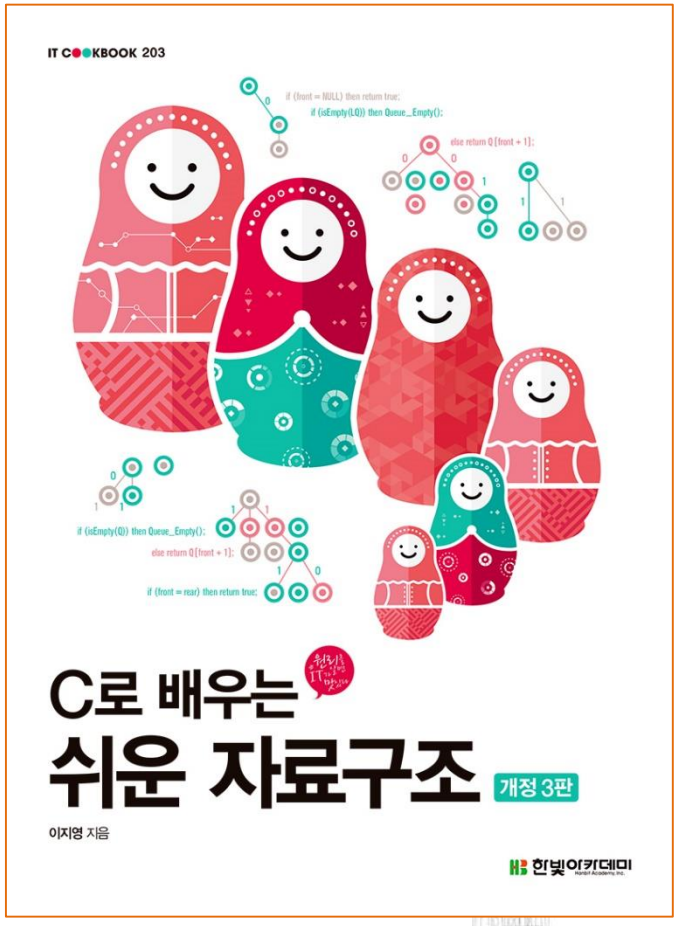
Easy data structure in C, 3rd edition, Jiyoung Lee

C로 배우는 쉬운 자료구조, 개정 3판

[강의교안 이용 안내]

Textbook

- 도서명 : C로 배우는 쉬운 자료구조, 개정 3판
- ISBN : 79-11-5664-269-5 93000
- 저자 : 이지영
- 출판사 : 한빛아카데미(주)
- 페이지 / 정가 : 596p / 27,000원
- 예제 소스 : http://www.hanbit.co.kr/exam/4269



2 자료구조 구현을 위한 C 프로그래밍 기법

IT CookBook, C로 배우는 쉬운 자료구조(개정 3판)

한빛아카데미 Hanbit Academy, Inc.

Contents

- Learning programming techniques to implement various data structures (algorithms) in C programs

- Arrays
- Pointers
- *Struct*
- *Recursive functions*

1. Arrays: definition

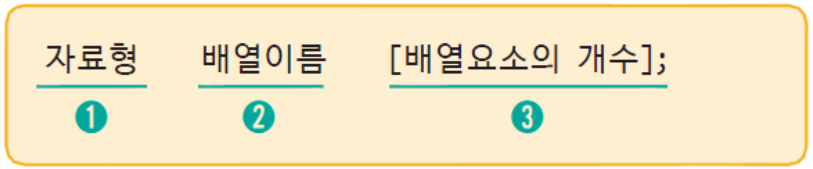
❖ Arrays

- Group of data elements with the same data types which are stored in the memory contiguously
- Index
  - Integer numbers to distinguish array items
  - Starts from 0 in C language
- All the data types in C can be arrays
- 1-dimensional, 2-dimensional, 3-dimensional, …, arrays

1. Arrays: 1-dimensional arrays

❖ Declaration of 1-dim arrays

- Type(1) Array_name(2) [Array_size](3);
  - (1) declares the base type of an array. All the elements should be the same type
  - (2) same as the variable naming rules
  - (3) Use brackets. The allocated memory size is Type_size * Array_size
    - In C, "`int A[10]`" → `sizeof(int)*10` .



❶ 배열의 자료형을 선언한다. 배열 요소는 모두 자료형이 같아야 하고, 배열 요소의 자료형이 배열의 자료형이 된다.

❷ 변수 이름과 같은 규칙으로 정한다.

❸ 대괄호([])를 사용해 배열 요소의 개수를 표시하는데, 배열 요소 개수가 배열 크기이다. 배열을 선언하면 메모리에 배열에 대한 공간이 할당되고 그 크기는 '자료형에 대한 메모리 할당 크기×배열 요소의 개수'이다.

그림 2-1 1차원 배열의 선언 형식

1. Arrays: 1-dimensional arrays

표 2-1 여러 자료형의 배열 선언 예와 의미

배열 선언 예	의미	배열 요소	메모리 할당 크기
char c[100];	char형 배열 요소 100개로 구성된 배열 c	c[0]~c[99]	1byte x 100
int i[100];	int형 배열 요소 100개로 구성된 배열 i	i[0]~i[99]	4byte x 100
short s[100];	short형 배열 요소 100개로 구성된 배열 s	s[0]~s[99]	2byte x 100
long l[100];	long형 배열 요소 100개로 구성된 배열 l	l[0]~l[99]	4byte x 100

The size of int type may vary with machine/OS/compilers

The size of int type may vary with machine/OS/compilers

```
int mid_score[40];
```

그림 2-2 배열 선언과 메모리 할당 구조 예 : 40명의 중간고사 점수

1. Arrays: sizes of various base types


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>

/* Get the sizes of different data types */
int main( void ) {
  char c[100];
  int i[100];
  short s[100];
  long l[100];
  float f[100];
  double d[100];

  printf("Type\tUnit bytes\tArray[100] bytes\n");
  printf("---------------------------------------------\n");
  printf("char\t%lu\t\t%lu\n",sizeof(char),sizeof(c));
  printf("int\t%lu\t\t%lu\n",sizeof(int),sizeof(i));
  printf("short\t%lu\t\t%lu\n",sizeof(short),sizeof(s));
  printf("long\t%lu\t\t%lu\n",sizeof(long),sizeof(l));
  printf("float\t%lu\t\t%lu\n",sizeof(float),sizeof(f));
  printf("double\t%lu\t\t%lu\n",sizeof(double),sizeof(d));
}
```

```
$ gcc -W -Wall sizes.c -o sizes.exe

$ ./sizes.exe
Type    Unit bytes      Array[100] bytes
---------------------------------------------
char    1               100
int     4               400
short   2               200
long    8               800
float   4               400
double  8               800
```

Array declaration and memory allocation

1. Arrays: sizes of various base types


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>
#include<stdint.h>

/* fixed-width integers, since C99 --- 1999 ISO standards
 * https://boycoding.tistory.com/151
 * https://en.cppreference.com/w/c/types/integer */

int main( void ) {
  int8_t i8[100];
  int16_t i16[100];
  int32_t i32[100];
  int64_t i64[100];

  printf("Type\tUnit bytes\tArray[100] bytes\n");
  printf("-----------------------------------------------\n");
  printf("int8_t\t%lu\t\t%lu\n",sizeof(int8_t),sizeof(i8));
  printf("int16_t\t%lu\t\t%lu\n",sizeof(int16_t),sizeof(i16));
  printf("int32_t\t%lu\t\t%lu\n",sizeof(int32_t),sizeof(i32));
  printf("int64_t\t%lu\t\t%lu\n",sizeof(int64_t),sizeof(i64));
}
```

```
$ gcc -W -Wall fixedint_sizes.c -o fixedint_sizes.exe

$ ./fixedint_sizes.exe
Type      Unit bytes      Array[100] bytes
-----------------------------------------------
int8_t    1               100
int16_t   2               200
int32_t   4               400
int64_t   8               800
```

1. Arrays: 1-dimensional arrays

❖ Initializing 1-dim arrays

- `TypeName ArrayName[ArraySize] = {ValueList};`
- Example

```
int A[5] = {1, 2, 3, 4, 5};
```

또는

```
int A[ ] = {1, 2, 3, 4, 5};
```

배열의 모든 원소에 초깃값을 주면
배열 크기 생략 가능

또는

```
int A[5];
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;
```

(a) 1차원 배열의 초기화

	A[0]	A[1]	A[2]	A[3]	A[4]
A	1	2	3	4	5

(b) 메모리 할당 구조

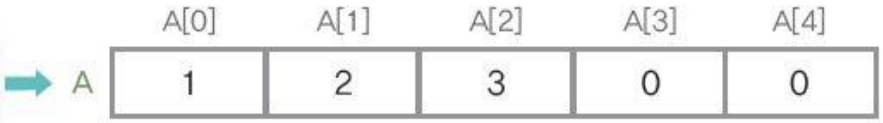
그림 2-4 1차원 배열의 초기화 예 1

❖ Initializing 1-dim arrays

```
int A[5] = {1, 2, 3};
```

또는

```
int A[5];
A[0] = 1;
A[1] = 2;
A[2] = 3;
```



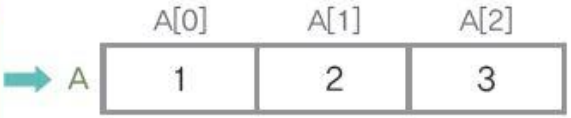
The number of initialization values < array size
➔ When initialization values are not given, the items are filled with 0 or random values, depending on machine/OS/compilers

(a) '초깃값의 개수 〈 배열 크기'인 경우

```
int A[3] = {1, 2, 3, 4, 5};
```

또는

```
int A[3];
A[0] = 1;
A[1] = 2;
A[2] = 3;
A[3] = 4;
A[4] = 5;
```



The number of initialization values > array size
➔ runtime error may occur
➔ Should be avoided

(b) '초깃값의 개수 〉 배열 크기'인 경우

그림 2-5 1차원 배열의 초기화 예 2

1-dimensional array initialization


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>
void print_array(int A[], int N, char name[]) {
  int i;
  printf("%s = [", name);
  for (i=0; i<N; i++) printf("%d ",A[i]);
  printf("]\n");
}

/* Array initialization examples */
int main( void ) {
  int A1[5] = {1,2,3,4,5};
  int A2[] = {1,2,3,4,5};
  int A3[5];
  int A4[5] = {1,2,3};

  print_array(A1, 5, "A1");
  print_array(A2, 5, "A2");
  print_array(A3, 5, "A3 org");
  A3[0] = 1; A3[1] = 2; A3[2] = 3;
  print_array(A3, 5, "A3 assigned");
  print_array(A4, 5, "A4");
}
```

```
$ gcc -W -Wall array_init_ex1.c -o
array_init_ex1.exe

$ ./array_init_ex1.exe
A1 = [1 2 3 4 5 ]
A2 = [1 2 3 4 5 ]
A3 org = [1 0 4196285 0 0 ]
A3 assigned = [1 2 3 0 0 ]
A4 = [1 2 3 0 0 ]
```

1-dimensional array initialization


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>
void print_array(int A[], int N, char name[]) {
  int i;
  printf("%s = [", name);
  for (i=0; i<N; i++) printf("%d ",A[i]);
  printf("]\n");
}

/* Array initialization examples */
int main( void ) {
  int A5[3] = {1,2,3,4,5};
  // int A6[];  // compilation error: array size missing in ‘A

  print_array(A5, 5, "A5");
}
```

```
$ gcc -W -Wall array_init_ex1.c -o
array_init_ex1.exe
$ gcc -W -Wall array_init_ex2.c -o
array_init_ex2.exe
array_init_ex2.c:18:22: warning:
excess elements in array initializer
   int A5[3] = {1,2,3,4,5};
                      ^
array_init_ex2.c:18:24: warning:
excess elements in array initializer
   int A5[3] = {1,2,3,4,5};
                        ^
$ ./array_init_ex2.exe
A5 = [1 2 3 0 2054084624 ]
```

Uninitialized slots are 0 and 2054084624 (undefined)

1. Arrays: char arrays for strings

❖ Character arrays

- Sequence of char data type values
- Expressed by double quotation marks (“…”)
- To store strings, char type arrays are used to represent sequence of characters
- Initialization
  - String literals: `char S[] = "Hello";`
  - Character arrays: `char S[] = {'H', 'e', 'l', 'l', 'o'};`

1. Arrays: char arrays for strings


```
char s1[10] = "String";
```

	s1[0]	s1[1]	s1[2]	s1[3]	s1[4]	s1[5]	s1[6]	s1[7]	s1[8]	s1[9]
s1	S	t	r	i	n	g	\0			

(a) 문자열을 사용한 초기화

```
char s2[10] = { 'S', 't', 'r', 'i', 'n', 'g' };
```

Wrong!!

	s2[0]	s2[1]	s2[2]	s2[3]	s2[4]	s2[5]	s2[6]	s2[7]	s2[8]	s2[9]
s2	S	t	r	i	n	g				

(b) 초깃값 문자 리스트를 사용한 초기화

그림 2-6 문자 배열의 선언과 메모리 할당 구조

```
char s1[] = "String";
```

➡

```
char s1[] = { 'S', 't', 'r', 'i', 'n', 'g', '\0' };
```

그림 2-7 문자열을 사용한 초기화 → 초깃값 문자 리스트를 사용한 초기화

1. Arrays: char array initialization


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>
#include<string.h>

/* String (char array) initialization examples */
int main( void ) {
  // String initialization --- 7 characters,
  // '\0' added at the end
  char S1[] = "String";
  // char array initialization --- exactly 6 characters
  char S2[] = {'S','t','r','i','n','g'};
  char S3[10] = "String";
  // char array initialization --- exactly 6 characters
  char S4[10] = {'S','t','r','i','n','g'};

  printf("sizeof(S1) = %lu, strlen(S1) = %lu, S1 = [%s]\n", sizeof(S1),strlen(S1),S1);
  printf("sizeof(S2) = %lu, strlen(S2) = %lu, S2 = [%s]\n", sizeof(S2),strlen(S2),S2);
  printf("sizeof(S3) = %lu, strlen(S3) = %lu, S3 = [%s]\n", sizeof(S3),strlen(S3),S3);
  printf("sizeof(S4) = %lu, strlen(S4) = %lu, S4 = [%s]\n", sizeof(S4),strlen(S4),S4);
}
```

```
$ gcc -W -Wall string_init_ex1.c -o string_init_ex1.exe

$ ./string_init_ex1.exe
sizeof(S1) = 7, strlen(S1) = 6, S1 = [String]
sizeof(S2) = 6, strlen(S2) = 6, S2 = [String]
sizeof(S3) = 10, strlen(S3) = 6, S3 = [String]
sizeof(S4) = 10, strlen(S4) = 6, S4 = [String]
```

Lucky!!
By chance at the ends of S2 and S4, 0 is filled. But not always

1. Arrays: Multidimensional arrays

❖ Multidimensional array declarations

- DataType ArrayName[Size1][Size2];
- DataType ArrayName[Size1][Size2][Size3];
- DataType ArrayName[Size1][Size2][Size3] …. [SizeN];

1. Arrays: 2-dimensional arrays

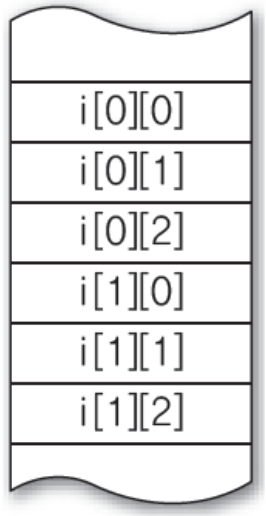

```
int i[2][3];
```

(a) 배열 선언

	열 번호 0	열 번호 1	열 번호 2
행 번호 0	i[0][0]	i[0][1]	i[0][2]
행 번호 1	i[1][0]	i[1][1]	i[1][2]

(b) 논리적 구조

Logical representation



(c) 물리적 구조

Physical representation

그림 2-10 2차원 배열의 선언과 논리적/물리적 구조 예

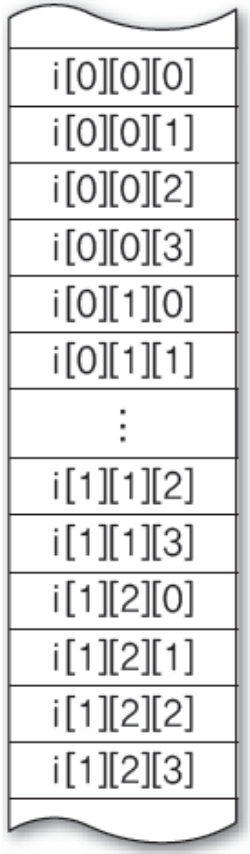
1. Arrays: 3-dimensional arrays


```
int i[2][3][4];
```

(a) 선언 형식

(b) 논리적 구조

Logical representation



(c) 물리적 구조

Physical representation

그림 2-12 3차원 배열을 선언한 예

1. Arrays: Multidimensional array initialization

- 2-dimensional array initialization

```
int i[2][3] = {{1, 2, 3}, {4, 5, 6}};    또는    int i[2][3] = {1, 2, 3, 4, 5, 6};
```

	[0]	[1]	[2]
[0]	1	2	3
[1]	4	5	6

그림 2-13 2차원 배열의 초기화와 논리적 구조

```
int i[][3] = {{1, 2, 3}, {4, 5, 6}};    또는    int i[][3] = {1, 2, 3, 4, 5, 6};
```

그림 2-14 행 크기를 생략한 2차원 배열의 초기화

Row sizes can be inferred automatically

1. Arrays: Multidimensional array initialization

- 3-dimensional array initialization

```
int i[2][3][4] = {{1, 2, 3, 4 }, {5, 6, 7, 8}, {9, 10, 11, 12},
                  {13, 14, 15, 16}, {17, 18, 19, 20}, {21, 22, 23, 24}};
```

면 1

13	14	15	16
17	18	19	20
21	22	23	24

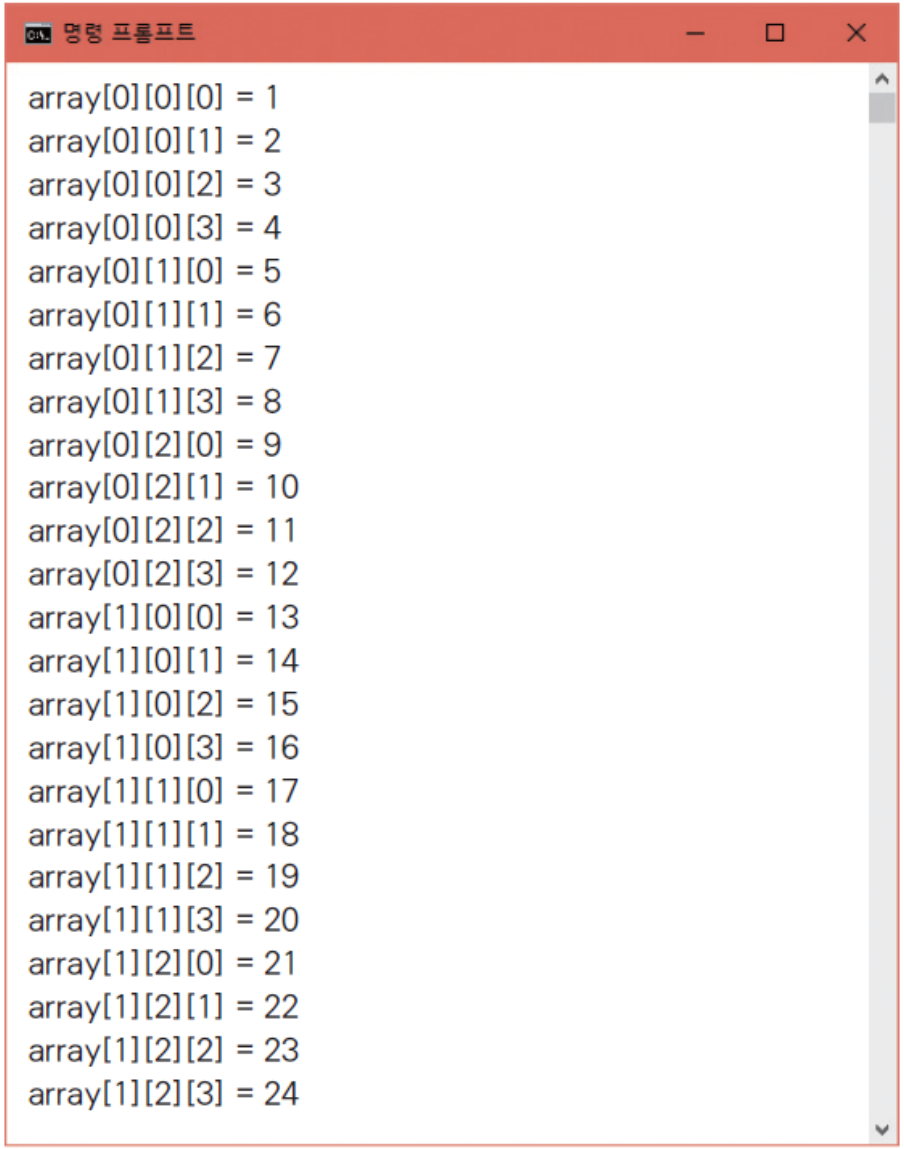
면 0

1	2	3	4
5	6	7	8
9	10	11	12

그림 2-15 3차원 배열의 초기화와 논리적 구조

1. Arrays: Multidimensional array initialization

- Example:
  3-dimensional array
  print



1. Arrays: Multidimensional arrays of strings

❖ 2-dim string array declaration

(a)

```
char c[3][20]={ "Hong Gil Dong",
                "Computer Department",
                "Seoul Korea"};
```

(b)

```
strcpy(c[0], "Hong Gil Dong");
strcpy(c[1], "Computer Department");
strcpy(c[2], "Seoul Korea");
```

```
char c[3][20];
```

그림 2-16 2차원 문자 배열의 선언 예

(c)

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]	[17]	[18]	[19]
[0]	H	o	n	g		G	i	l		D	o	n	g	\0						
[1]	C	o	m	p	u	t	e	r		D	e	p	a	r	t	m	e	n	t	\0
[2]	S	e	o	u	l		K	o	r	e	a	\0								

그림 2-17 2차원 문자 배열의 문자열 저장과 논리적 구조

1. Arrays: Multidimensional arrays of strings


```
/* ID: COMP319
 * NAME: Algorithms 1
 * OS: linux, Ubuntu 16.04
 * Compiler version: gcc 5.4.0 20160609 */

#include<stdio.h>
#include<string.h>       // for strlen and strcpy

// maximum string length
#define MAX_STR      128

/* String array (2-dim char array) examples */
int main( void ) {
  char c[3][MAX_STR];

  strcpy(c[0],"COMP319");
  strcpy(c[1],"Algorithms 1");
  strcpy(c[2],"Electronics Engineering");

  printf("ID: %s\n",c[0]);
  printf("NAME: %s\n",c[1]);
  printf("DEPARTMENT: %s\n",c[2]);
}
```

```
$ gcc -W -Wall string_array_ex.c -o string_array_ex.exe

$ ./string_array_ex.exe
ID: COMP319
NAME: Algorithms 1
DEPARTMENT: Electronics Engineering
```

Pointers

2. Pointers: definition

❖ What are pointers?

- Memory addresses of variables

❖ What are pointer variables?

- Variables to store pointers (memory addresses)
- Pointer variable POINTS to a variable

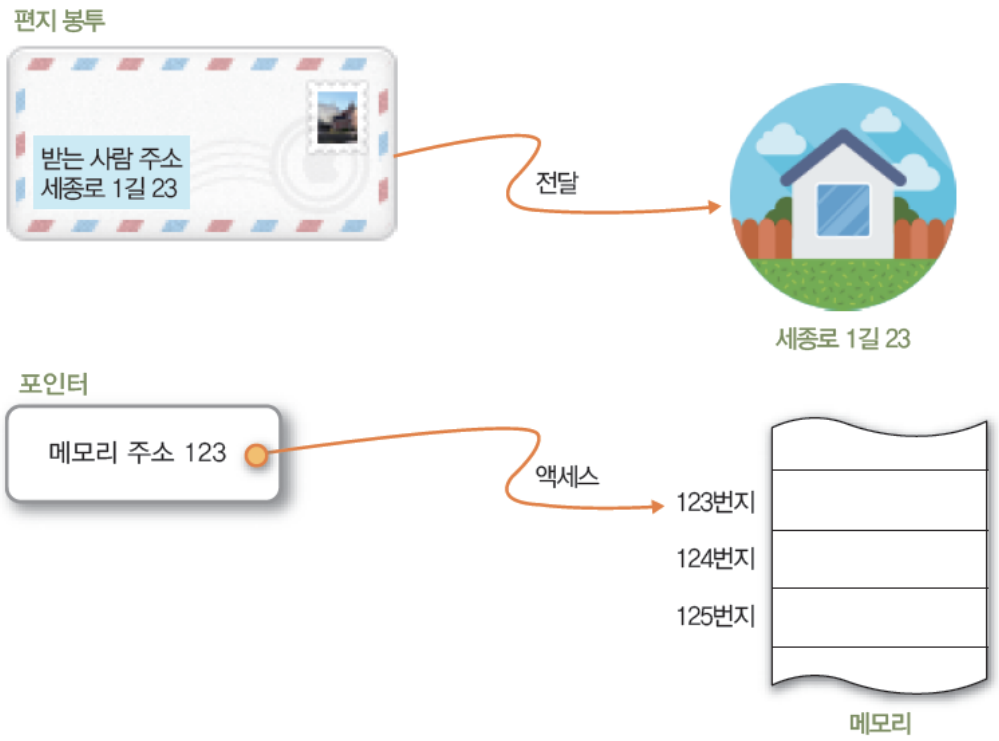


그림 2-18 포인터의 액세스 처리

2. Pointers: declaration

❖ Pointer declaration format

- `TypeName *PointerName;`
  - Type of the variable to be indicated
  - The operator `*` indicates that `PointerName` is a pointer variable

```
int i;
int *ptr = &i;
```

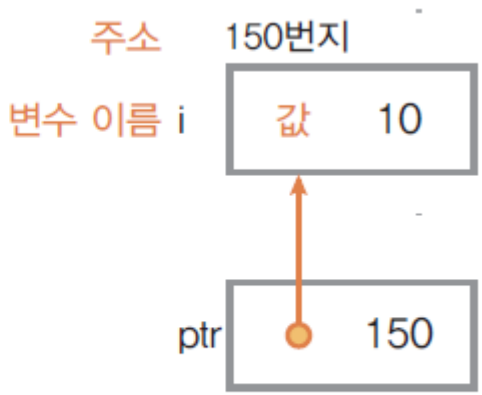


그림 2-19 포인터의 사용 예

2. Pointers: declaration

❖ Declaring pointers of various types

```
char *ptr;
```
❶ 1바이트의 char형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 1바이트의 char 데이터를 액세스한다.

```
short *ptr;
```
❷ 2바이트의 short형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 2바이트의 short 데이터를 액세스한다.

```
int *ptr;
```
❸ 4바이트의 int형 변수의 주소를 저장할 포인터를 선언한 예이다.
ptr에 저장된 메모리 주소로부터 4바이트의 int 데이터를 액세스한다.

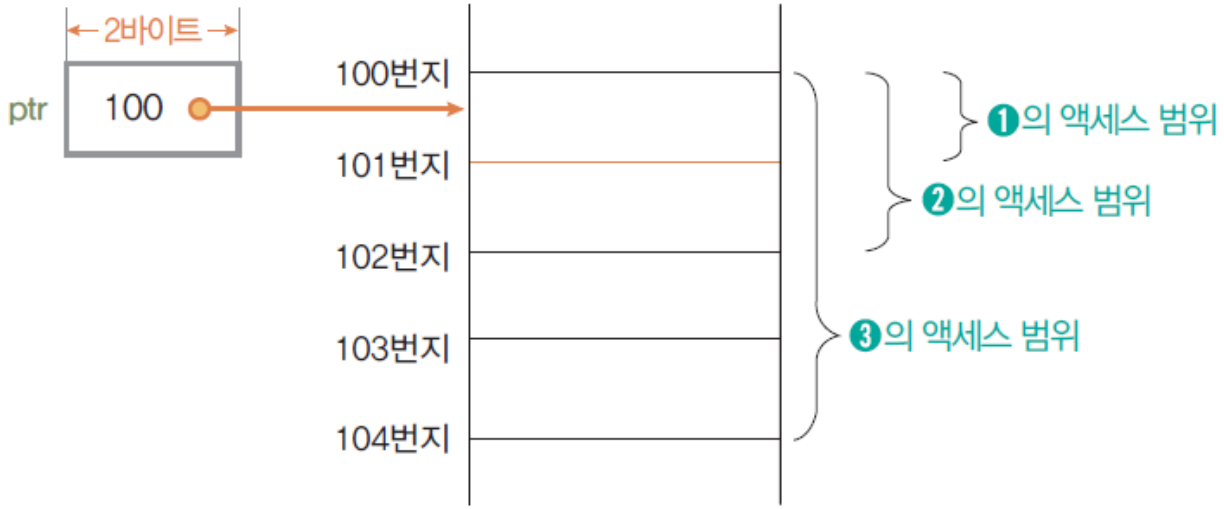


그림 2-21 포인터에서 선언한 자료형에 따른 메모리 액세스 범위

2. Pointers: operators

❖ Dereference (address-of) operator: &

- Used to obtain the memory address of a variable

- `PointerVariable = &Variable;`

(a)

```
int i = 10;
int *ptr;
```

(b)

(c)

```
ptr = &i;
```

(d)

그림 2-23 포인터 선언과 사용 예

2. Pointers: operators

❖ Pointer indirection operator: *

- Used to obtain the variable pointed by a pointer

- `*PointerVariable = Value;`
  - // equivalent to Variable = Value
- `Variable = *PointerVariable;`

2. Pointers: operators

Caution: the meaning of [*] is different in pointer variable declaration and dereferencing

```
int i, j;
int *ptr;
ptr = &i;
```

❶ 주소 연산자를 사용하여 변수 i의 주소를 포인터 ptr에 할당한다. 포인터 ptr은 변수 i를 가리킨다.

```
*ptr = 10;
```

❷ 참조 연산자를 사용하여 포인터 ptr이 가리키는 영역에 값 10을 지정한다. 따라서 변수 i에는 10이 저장된다.

```
j = *ptr;
```

❸ 다시 참조 연산자를 사용하여 ptr이 가리키는 영역의 값을 변수 j에 지정한다. 따라서 ptr이 가리키는 변수 i의 값인 10을 변수 j에 저장한다.

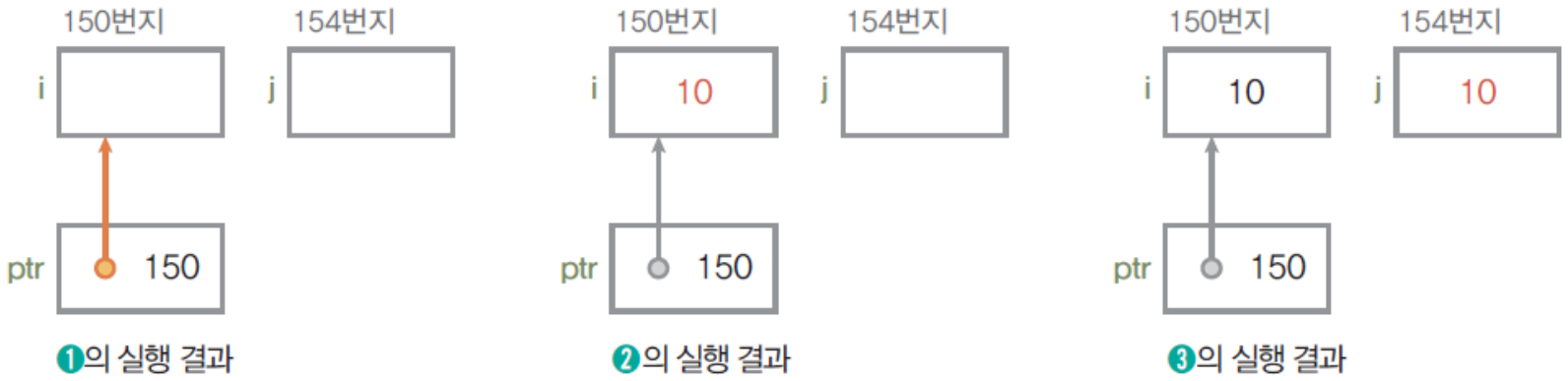


그림 2-25 포인터 연산자를 사용한 예

2. Pointers: variable access example

ex2_8.c

```c
#include <stdio.h>
int main()
{
  int i=10, j=20;
  int *ptr;

  printf("The value of i = %d₩n", i);
  printf("The value of j = %d₩n", j);
  printf("The address of i (&i) = %ld₩n",(long)(&i));
  printf("The address of j (&j) = %ld₩n",(long)(&j));

  ptr = &i;
  printf("ptr after <<ptr = &i>> = %ld₩n",(long)ptr);
  printf("*ptr = %d₩n", *ptr);

  ptr = &j;
  printf("ptr after <<ptr = &j>> = %ld₩n",(long)ptr);
  printf("*ptr = %d₩n", *ptr);

  i = *ptr;
  printf("Address of i after <<i=*ptr>> = %ld₩n",
      (long)(&i));
  printf("Value of i = %d₩n", i);
}
```

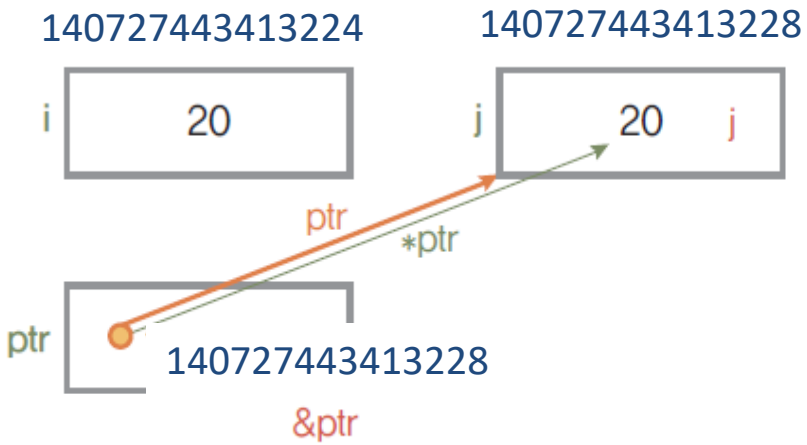


그림 2-26 [예제 2-8]을 실행한 결과

```
$ gcc -W -Wall ex2_8.c -o ex2_8.exe
$ ./ex2_8.exe
/ex2_8.exe
The value of i = 10
The value of j = 20
The address of i (&i) = 140727443413224
The address of j (&j) = 140727443413228
ptr after <<ptr = &i>> = 140727443413224
*ptr = 10
ptr after <<ptr = &j>> = 140727443413228
*ptr = 20
Address of i after <<i=*ptr>> = 140727443413224
Value of i = 20
```

2. Pointers: initialization

❖ Pointer initialization methods

❶ Using address-of operators

```
int i;
int *ptr = &i;
```

❷ Using dynamic memory allocation

```
char *ptr = (char *)malloc(100);
```

❸ From arrays

```
char A[100];
char *ptr = A;
```

```
char A[100];
char *ptr = &A[0];
```

[char *ptr = &(A[0]);] is better to avoid ambiguity

2. Pointers: operators

ex2_9.c

```
1  #include <stdio.h>
2  #include <string.h>     // for strdup()
3  #include <stdlib.h>     // for free()
4  int main() {
5    char string1[] = "Dreams come true!";
6    char *ptr1, *ptr2;
7    int i;
8
9    ptr1 = string1;
10   ptr2 = strdup(string1);
11
12   printf("string1: %ld, [%s]\n", (long)string1, string1);
13   printf("ptr1: %ld, [%s]\n", (long)ptr1, ptr1);
14   printf("ptr2: %ld, [%s]\n", (long)ptr2, ptr2);
15
16   printf("string1 <-- \"Peaces come true!\"\n");
17   strncpy(string1,"Peaces",6);
18   printf("string1: %ld, [%s]\n", (long)string1, string1);
19   printf("ptr1: %ld, [%s]\n", (long)ptr1, ptr1);
20   printf("ptr2: %ld, [%s]\n", (long)ptr2, ptr2);
21
22   printf("<Reverse printing>\n");
23   printf("rev(ptr1) = [");
24   for (i=strlen(ptr1)-1; i>=0; i--) printf("%c",ptr1[i]);
25   printf("]\nrev(ptr2) = [");
26   for (i=strlen(ptr2)-1; i>=0; i--) printf("%c",ptr2[i]);
27   printf("]\n");
28
29   /* note: string generated by strdup should be "free()'d"
30    * because they are permanently allocated */
31   free(ptr2); printf("ptr2 is freed\n");
32   /* but not ptr1 -- causes memory error, should be commented out */
33   free(ptr1); printf("ptr1 is freed\n");
34 }
```

```
$ gcc -W -Wall ex2_9.c -o ex2_9.exe
$ ./ex2_9.exe
string1: 140735814800544,
   [Dreams come true!]
ptr1: 140735814800544,
   [Dreams come true!]
ptr2: 94898437653088,
   [Dreams come true!]
string1 <-- "Peaces come true!"
string1: 140735814800544,
   [Peaces come true!]
ptr1: 140735814800544,
   [Peaces come true!]
ptr2: 94898437653088,
   [Dreams come true!]
<Reverse printing>
rev(ptr1) = [!eurt emoc secaeP]
rev(ptr2) = [!eurt emoc smaerD]
ptr2 is freed
double free or corruption (out)
terminated (core dumped)
```

Intentionally added to show the erroneous case

2. Pointers: pointer arrays

❖ Pointer array definition

- Arrays of pointer variables
- `TypeName *PointerArrayName[ArraySize];`

2. Pointers: pointer arrays

❖ Comparison of 2-dim arrays and pointer arrays

(a) `char string[3][10] = { "Dreams", "come", "true!" };`

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
string[0]	D	r	e	a	m	s	\0			
string[1]	c	o	m	e	\0					
string[2]	t	r	u	e	!	\0				

(b) `char *ptr[3] = { { "Dreams" },{ "come" },{ "true!" } };`

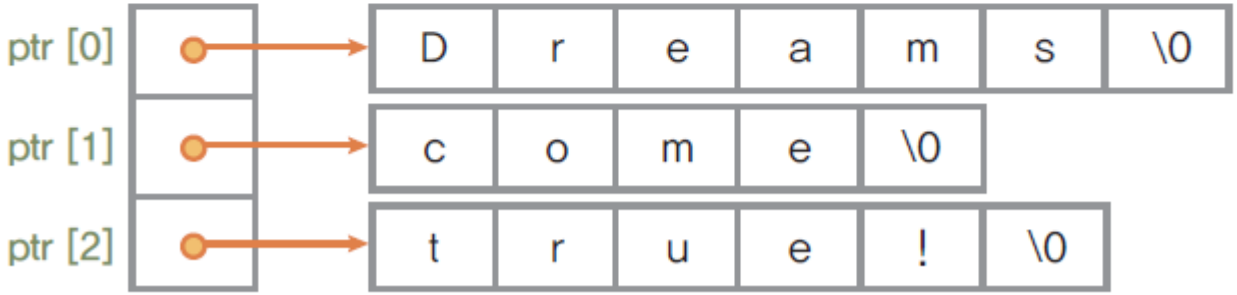


그림 2-28 2차원 배열과 1차원 포인터 배열의 비교

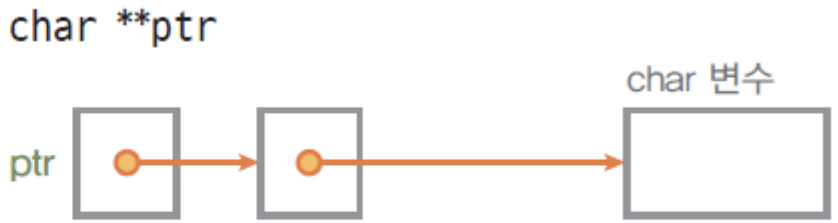
2. Pointers: cascaded

❖ Pointer of a pointer

- Double pointers. A pointer to a pointer variable.

```
자료형 **포인터이름;
```

(a) 선언 형식



(b) 사용 예

그림 2-29 포인터의 포인터 선언 형식과 예

Pointer arrays of strings

ex2_10.c

```c
#include <stdio.h>
#include <string.h>     // strdup()
#include <stdlib.h>     // malloc(), free()
int main()
{
  char string[3][10] = {"Dreams","come","true!"};
  char *ptr1[3], **ptr2;
  int i;

  printf("string: [");
  for (i=0; i<3; i++) printf("%s ",string[i]);
  printf("]\n");

  for (i=0; i<3; i++) ptr1[i] = strdup(string[i]);
  printf("ptr1: [");
  for (i=0; i<3; i++) printf("%s ",ptr1[i]);
  printf("]\n");

  ptr2 = (char**)malloc(sizeof(char*)*3);
  for (i=0; i<3; i++) ptr2[i] = strdup(string[i]);
  printf("ptr2: [");
  for (i=0; i<3; i++) printf("%s ",ptr2[i]);
  printf("]\n");

  /* note: string generated by strdup should be "free()'d"
   * because they are permanently allocated */
  for (i=0; i<3; i++) free(ptr1[i]);
  printf("ptr1 is freed\n");
  for (i=0; i<3; i++) free(ptr2[i]);
  free(ptr2);
  printf("ptr2 is freed\n");

  /* but not local string array string -- causes memory error
   * they are returned to memory automatically after the function ends */
  for (i=0; i<3; i++) free(string[i]);
  printf("string is freed\n");
}
```

```
$ gcc -W -Wall ex2_10.c -o ex2_10.exe
$ ./ex2_10.exe
string: [Dreams come true! ]
ptr1: [Dreams come true! ]
ptr2: [Dreams come true! ]
ptr1 is freed
ptr2 is freed
munmap_chunk(): invalid pointer
terminated (core dumped)
```

Intentionally added to show the erroneous case

Thank You !

IT CookBook, C로 배우는 쉬운 자료구조(개정판)

한빛미디어 Hanbit Media, Inc.