학번 : _____ 성명 : _____.

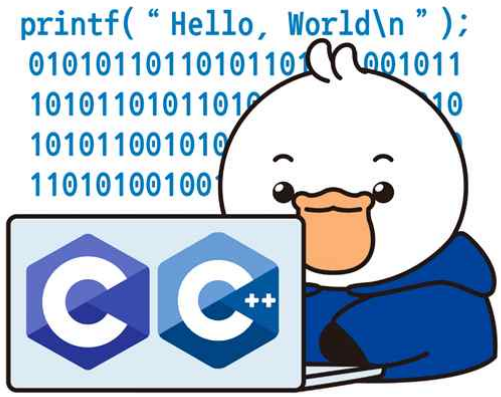
- 본 강의자료는 과학기술정보통신부 및 정보통신기획평가원에서 지원하는『소프트웨어중심대학』사업의 결과물입니다.
- 본 강의자료는 내용은 전재할 수 없으며, 인용할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원의 '소프트웨어중심대학'의 결과물이라는 출처를 밝혀야 합니다.

SW 대구가톨릭대학교 소프트웨어중심대학사업단

대구가톨릭대학교 컴퓨터소프트웨어학부 School of Computer Software, Daegu Catholic University

Part 7. 포인터

목차



7.1 메모리와 주소

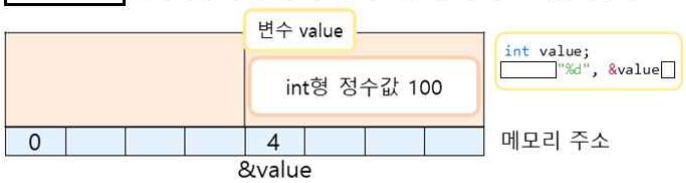
- [ ]와 [ ] 메모리 · 값

  - ✓ 컴퓨터에서 실행되는 모든 소프트웨어는 [ ]에 적재 — 메모리
    - ➢ [ ]에 저장되지 않은 것은 어떠한 경우에도 실행 불가능 — 메모리
  - ✓ 메모리 내 위치 식별을 위해 1[ ]단위로 [ ] 할당 — 바이트 · 값
    - ➢ 시작 위치에 0번 주소, 이후 1[ ]당 1씩 증가하는 [ ] 할당 — 바이트 · 값
  - ✓ C언어에서는 변수의 이름을 이용하지만, 실제 동작 단계에서는 변수 이름이 아닌, 변수의 [ ]를 통해 변수 접근 — 주소
    - ➢ '우리집' 이라는 변수는 누가 사용하는지에 따라 [ ]가 변경 — 주소

      Ex) 홍길동의 '우리집'과 유관순의 '우리집'은 실제로 다른 위치에 존재

7.1 메모리와 주소

- 주소 연산자 & (앰퍼센드)
  - ✓ 지금까지의 프로그램 작성에서 대부분의 경우 변수를 통해 값 저장 및 접근
    - ➢ [ ]함수를 사용하여 값을 입력받는 경우에만 주소 연산자 사용 scanf()
  - ✓ 전위 단항 연산자로, 두 번째로 높은 우선순위를 가지는 연산자
  - ✓ 오직 변수에만 사용 가능 → 고정된 값 상수는 주소가 존재하지 않음
  - ✓ 변수의 데이터를 저장하기 위한 메모리 상의 주소를 확인하기 위해 사용
    - ➢ [ ]함수를 사용하여 정수 형태의 주소 값 출력 printf()



7.1 메모리와 주소

소스 7-1

```
 1 #include <stdio.h>
 2 int main(void)
 3 {
 4    int n;
 5    printf("Input an integer : ");
 6    scanf("%d", &n);
 7    printf("Value : %d\n", n);
 8    printf("Address : %u, %p, %#x, %#X\n", &n, &n, &n, &n);
 9    printf("Size of address : %d\n", sizeof(&n));
10    printf("Size of value : %d\n", sizeof(n));
11    return 0;
12 }
```

입력 예1
```
50
```

출력 예1
```
Input Integer : 50
Value : 50
Address : 2142799196, 0x7ffd7fb8855c, 0x7fb8855c, 0X7FB8855C
Size of Address : 8
Size of value : 4
```

7.1 메모리와 주소

입력 예2	150
출력 예2	Input Integer : 150 Value : 150 Address : 2102140492, 0x7ffc7d4c1e4c, 0x7d4c1e4c, 0X7D4C1E4C Size of address : 8 Size of value : 4

7.2 포인터

- [ ] 변수와 메모리 주소 저장 포인터

✓ 모든 변수는 메모리로부터 공간을 할당 받기 때문에 주소를 가짐

➢ 주소도 하나의 값이기 때문에 메모리에 저장 가능

✓ 따라서 주소를 메모리에 저장하기 위해서는 메모리 할당이 필요하고,
메모리 할당을 위해서는 변수 선언 필요

✓ [ ] : 메모리 주소를 저장하기 위한 변수 포인터

➢ 다른 변수의 앞에 주소 연산자 [ ]를 붙여 얻은 주소 저장 &

7.2 포인터

- 포인터 변수 선언

  ✓ 포인터 변수 선언을 위해서는 간접 연산자 *를 변수 앞에 사용

  ➢ 간접 연산자 *또한 전위 단항 연산자로, 두 번째 우선순위 가짐

  ➢ 산술 곱셈 연산자와 동일한 기호를 가지지만, 간접 연산자는 단항 연산자, 곱셈 연산자는 이항 연산자이므로 피 연산자 개수로 구분 가능

  ✓ 각 데이터 형에 따라 아래와 같이 포인터 변수를 선언 가능

  ✓ 또한 포인터 변수는 모두 8바이트( ) 또는 4바이트( )로 구성

데이터타입 *변수명;	int *ptrint; char *ptrchar; double *ptrdouble; float *ptrfloat;	int*ptrint; char * ptrchar; double*ptrdouble; float * ptrfloat;

< 각 데이터 형에 따른 포인터 변수 선언 예시 >

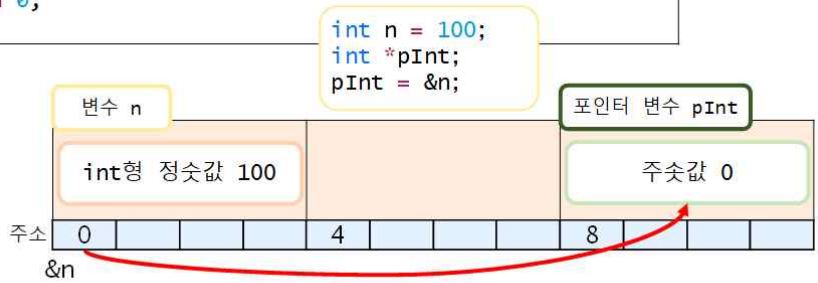
7.2 포인터

소스 7-2

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int n = 10;
5     int *pInt = &n;
6     printf("Address of i : %lu %p\n", &n, &n);
7     printf("Value of pInt : %lu %p\n", pInt, pInt);
8     printf("Address of pInt : %lu %p\n", &pInt, &pInt);
9     return 0;
10 }
```

출력 예

```
Address of i : 140736100918856 0x7fffad4d6648
Value of pInt : 140736100918856 0x7fffad4d6648
Address of pInt : 140736100918848 0x7fffad4d6640
```



7.2 포인터

소스 7-3

```
1  #include <stdio.h>
2  int main(void){
3     int n1 = 100, n2 = 50, *ptr;
4     ptr = &n1;
5     printf("Address of n1 : %p\n", &n1);
6     printf("Value of ptr : %p\n", ptr);
7     printf("Address of ptr : %p\n", &ptr);
8     ptr = &n2;
9     printf("Address of n2 : %p\n", &n2);
10    printf("Value of ptr : %p\n", ptr);
11    printf("Address of ptr : %p\n", &ptr);
12    return 0;
13 }
```

출력 예

```
Address of n1 : 0x7ffe8446452c
Value of ptr : 0x7ffe8446452c
Address of ptr : 0x7ffe84464520
Address of n2 : 0x7ffe84464528
Value of ptr : 0x7ffe84464528
Address of ptr : 0x7ffe84464520
```


7.2 포인터

- 간접 연산자 *
  - ✓ 포인터 변수는 포인터가 가리키는 변수의 주소 값을 저장
    - ➢ 저장된 주소를 이용하여 포인터 변수도 해당 주소에 저장된 값에 접근 가능
    - ➢ 포인터 변수가 가리키는 변수를 [ ]하려면 간접 연산자 * 를 이용
  - ✓ 아래는 간접 연산자를 통해 포인터가 가리키는 곳에 저장된 값을 [ ]하는 코드

참조 참조

```
int value = 100;
int *ptr; //포인터 변수 선언
ptr = &value;
printf("Dereference : %d \n", *ptr); //간접연산자 사용
```

7.2 포인터

소스 7-4

```
 1 #include <stdio.h>
 2 int main(void)
 3 {
 4    int value = 100;
 5    int *ptr = &value;
 6    printf("value : %d\n", value);
 7    printf("*ptr : %d\n", *ptr);
 8    *ptr = 200;
 9    printf("value : %d\n", value);
10    return 0;
11 }
```

출력 예

```
value : 100
*ptr : 100
value : 200
```


7.2 포인터

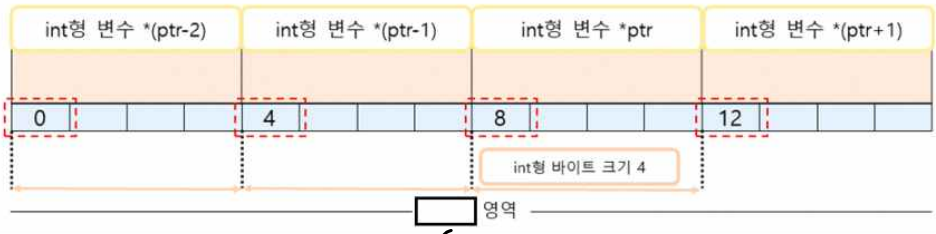
- 포인터 변수의 주소 연산

메모리

  - ✓ 포인터 변수에 간단한 덧셈, 뺄셈을 통해 이웃한 [ ] 주소에 접근할 수 있다
  - ✓ int형 포인터 변수 ptr에서 *(ptr + 1), *(ptr - 1)을 통해 이전, 다음 주소 접근
    - ➢ 포인터 주소 연산의 경우 포인터가 가리키는 변수의 [ ] 크기에 비례

데이터타입

    - ➢ ptr의 주소값이 0x0100인 경우, *(ptr + 1)은 0x0101이 아니라 0x0104이다
    - ➢ ptr의 주소값이 0x0100인 경우, *(ptr - 1)은 0x00ff가 아니라 0x00fc이다



주소 (메모리)

7.2 포인터

소스 7-5

```
1  #include <stdio.h>
2  int main(void)
3  {
4     int *pInt = 0x0000;
5     double *pDbl = 0x0000;
6     char *pChar = 0x0000;
7     printf("%p %p %p\n", pInt, pDbl, pChar);
8     printf("%p %p\n", pInt+1, pInt+2);
9     printf("%p %p\n", pDbl+1, pDbl+2);
10    printf("%p %p\n", pChar+1, pChar+2);
11    return 0;
12 }
```

입력 예	없음
출력 예	(nil) (nil) (nil) 0x4 0x8 0x8 0x10 0x1 0x2

7.2 포인터

- 포인터 [ ] 형변환
  - ✓ 메모리는 1바이트 단위로 주소가 부여되어 있음
    - ➢ 따라서 데이터를 읽고 쓰는 작업은 바이트 단위로 이뤄짐
  - ✓ 포인터 변수는 데이터 타입을 가지고 있기 때문에 int형 포인터 변수가 지시하는 곳은 다른 자료형 포인터로 읽기 불가
  - ✓ 그러나 동일한 주소를 다른 데이터 타입의 포인터를 사용하여 읽을 수 있음
    - ➢ int형 데이터가 저장된 주소를 char 포인터 변수가 가리키게 하기 위해 변수 또는 상수의 데이터 타입을 변경하는 [ ](cast)을 사용 형변환
  - ✓ 다음은 정수형 변수의 주소를 char형 포인터 변수가 가리키게 하는 코드

7.2 포인터

소스 7-6

```
1 #include <stdio.h>
2 int main(void){
3    int value = 0x12345678;
4    int *pi = &value;
5    char *pc = &value;
6    return 0;
7 }
```

컴파일 메시지

```
main.c:5:8: warning: incompatible pointer types
      initializing 'char *' with an expression of type
      'int *' [-Wincompatible-pointer-types]
        char *pc = &value;
              ^    ~~~~~~
1 warning generated.
```


7.2 포인터

소스 7-7

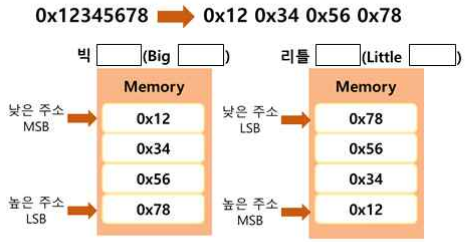
```
1 #include <stdio.h>
2 int main(void){
3    int hex = 0x12345678, i;
4    char* ptr = (char*)&hex;
5    for(i = 0; i < sizeof(int); i++){
6       printf("%0#6x\n", *(ptr+i));
7    }
8    return 0;
9 }
```

출력 예

```
0x0078
0x0056
0x0034
0x0012
```

7.2 포인터

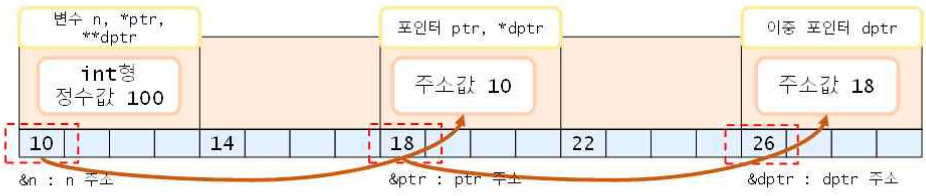
- 리틀 [ ], 빅 [ ]
  - ✓ 빅 [ ]
    - ➢ 낮은 주소 위치에 최상위 바이트(Most Significant Bit)부터 저장
  - ✓ 리틀 [ ]
    - ➢ 낮은 주소 위치에 최하위 바이트(Least Significant Bit)부터 저장



7.3 다중 포인터

- 다중 포인터
  - ✓ 포인터 변수도 [ ]이기 때문에 다른 포인터 변수의 [ ]를 가리킬 수 있다
    - ➢ 따라서 포인터에 대한 포인터 또는 포인터의 포인터가 가능하다
      - 이를 다중 포인터라 하며, 포인터 변수 개수에 따라 이중, 삼중이 된다
    - ➢ [ ] 포인터 변수 선언을 위해서는 간접 연산자를 2개 사용하여야 한다

```
int n = 100;
int *ptr = &n;
int **dptr = &ptr;
```



7.3 다중 포인터

소스 7-8

```c
#include <stdio.h>
int main(void){
    int number = 3;
    int *ptr = &number;
    int **dptr = &ptr;
    int ***tptr = &dptr;
    printf("number : %d\n", number);
    printf("*ptr : %d\n", *ptr);
    printf("**dptr : %d\n", **dptr);
    printf("***tptr : %d\n\n", ***tptr);
    printf("&number : %p\n", &number);
    printf("ptr : %p\n", ptr);
    printf("*dptr : %p\n", *dptr);
    printf("**tptr : %p\n\n", **tptr);
    printf("&ptr : %p\n", &ptr);
    printf("dptr: %p\n", dptr);
    printf("*tptr: %p\n", *tptr);
    printf("&dptr: %p\n", &dptr);
    printf("tptr: %p\n", tptr);
    return 0;
}
```

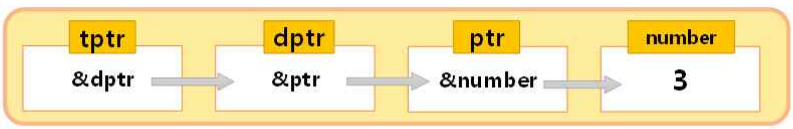
출력 예

```
number : 3
*ptr : 3
**dptr : 3
***tptr : 3
&number : 0x7ffc80793b54
ptr : 0x7ffc80793b54
*dptr : 0x7ffc80793b54
**tptr : 0x7ffc80793b54
&ptr : 0x7ffc80793b48
dptr : 0x7ffc80793b48
*tptr : 0x7ffc80793b48
&dptr : 0x7ffc80793b40
Tptr : 0x7ffc80793b40
```


7.3 다중 포인터

- 다중 포인터
  - ✓ 소스 7-8의 int형 변수 number, 포인터 변수 ptr, 이중 포인터 변수 dptr, 삼중 포인터 변수 tptr 모두 변수 [ ]의 값에 접근할 수 있다
  - ✓ 아래 그림은 모든 변수들의 관계를 나타낸다

number



7.3 다중 포인터

소스 7-9

```
1  #include <stdio.h>
2  int main(void) {
3     float f =30.0;
4     float *pf =&f;
5     float **dpf =&pf;
6     printf("%.1f %.1f %.1f \n", f, *pf, **dpf);
7     *pf = f *10; //fvalue = fvalue * 2;
8     printf("%.1f %.1f %.1f \n", f, *pf, **dpf);
9     **dpf =*pf *10; //fvalue = fvalue * 2;
10    printf("%.1f %.1f %.1f \n", f, *pf, **dpf);
11    return 0;
12 }
```

출력 예

```
30.0 30.0 30.0
300.0 300.0 300.0
3000.0 3000.0 3000.0
```


7.4 포인터와 배열

- 배열 이름과 포인터

  ✓ 배열의 이름은 배열이 저장된 [ ]를 가지는 상수이다

  ➢ 배열 이름, 포인터 변수 모두 [ ]를 가리킴

  ➢ 아래와 같이 배열 원소 값, 주소에 배열 이름을 통해 접근 가능

  ✓ 배열 age를 [ ] 변수라고 생각하면 아래 표를 더 쉽게 이해할 수 있다

배열 원소값		15	26	37
원소의 값 접근	age[i]	age[0]	age[1]	age[2]
	*(age + i)	*age	*(age+1)	*(age+2)
원소의 [ ] 접근	&age[i]	&age[0]	&(age[1])	&(age[2])
	age+i	age	(age+1)	(age+2)

7.4 포인터와 배열

소스 7-10

```
1  #include <stdio.h>
2  int main(void) {
3     int age[3] = {15, 26, 37};
4     printf("%d\n", age);
5     printf("%p\n", age);
6     printf("%p\n", &age[0]);
7  }
```

출력 예

```
-1904233964
0x7fff8e7fb214
0x7fff8e7fb214
```


7.4 포인터와 배열

소스 7-11

```
1  #include <stdio.h>
2  #define SIZE 3
3  int main(void)
4  {
5     int age[SIZE] = {15, 26, 37}, i ;
6     printf("age : %u, &age[0]: %u \n", age, &age[0]);
7     printf("age : %p, &age[0]: %p \n", age, &age[0]);
8     printf("index\taddress(%%u)\t address(%%p)\tvalue\n");
9     for(i=0; i<SIZE; i++)
10       printf("%5d\t%101\t%12p\t%5d\n", i, (age+i), (age+i), *(age+i));
11    return 0;
12 }
```

입력 예: 해당없음

출력 예

```
age : 1003440048, &age[0]: 1003440048
age : 0x7fff3bcf47b0, &age[0]: 0x7fff3bcf47b0
index    address(%u)                 address(%p)          value
    0     1003440048              0x7fff3bcf47b0             15
    1     1003440052              0x7fff3bcf47b4             26
    2     1003440056              0x7fff3bcf47b8             37
```

7.4 포인터와 배열

소스 7-12

```c
#include <stdio.h>
int main(void)
{
   double arr[5] = {1.1, 2.2, 3.3, 4.4, 5.5};
   int i;
   double *ptr = arr;
   for(i=0; i<5; i++)
      printf("[%d] %.2f %.2f %.2f %.2f\n", i, *(ptr+i), ptr[i], *(arr+i), arr[i]);
   return 0;
}
```

출력 예

```
[0] 1.10 1.10 1.10 1.10
[1] 2.20 2.20 2.20 2.20
[2] 3.30 3.30 3.30 3.30
[3] 4.40 4.40 4.40 4.40
[4] 5.50 5.50 5.50 5.50
```


7.4 포인터와 배열

- 포인터 변수와 증감 연산자
  - ✓ 배열을 가리키는 포인터 변수를 통해 배열의 각 원소에 접근 시, 증감 연산자를 통해 접근 가능
  - ✓ 배열 이름은 포인터 상수라서 값 변경이 불가능
    - ➢ 배열 이름을 가리키는 포인터 변수를 사용하여 배열 각 원소에 접근

7.4 포인터와 배열

소스 7-13

```
 1 #include <stdio.h>
 2 int main(void) {
 3    int num[3] = {10, 20, 30};
 4    int *ptr = num;
 5    printf("*ptr++ : %d \n", *ptr++);
 6    printf("*ptr : %d \n", *ptr);
 7    printf(" ++(*ptr) : %d \n", ++(*ptr));
 8    printf("(*ptr)++ : %d \n", (*ptr)++);
 9    printf("*ptr : %d \n", *ptr);
10    printf("*ptr+1 : %d \n", *ptr+1);
11    printf("*(ptr+1) : %d \n", *(ptr+1));
12    return 0;
13 }
```

출력 예

```
*ptr++ : 10
*ptr : 20
 ++(*ptr) : 21
(*ptr)++ : 21
*ptr : 22
*ptr+1 : 23
*(ptr+1) : 30
```

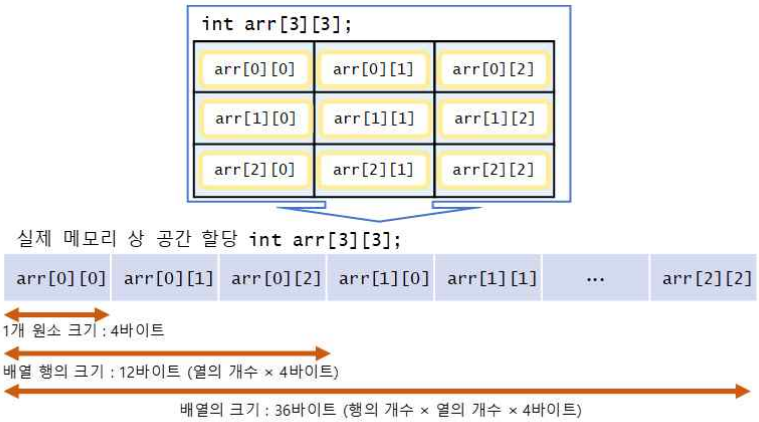

7.4 포인터와 배열

- 다차원 배열과 포인터
  - ✓ 2차원, 3차원 배열 또한 마찬가지로 포인터 변수를 통해 접근 가능
    - ➢ 2차원 배열은 1차원 배열을 원소로 가지는 다차원 배열
    - ➢ 3차원 배열은 2차원 배열을 원소로 가지는 다차원 배열
    - ➢ 따라서 2차원 배열은 □중 포인터, 3차원 배열은 □중 포인터 변수를 통해 각 배열의 원소에 접근 가능

2중 (3중

7.4 포인터와 배열

- 다차원 배열과 포인터
  - ✓ 2차원 배열이 메모리에 저장되는 경우 아래와 같이 저장



7.4 포인터와 배열

소스 7-14

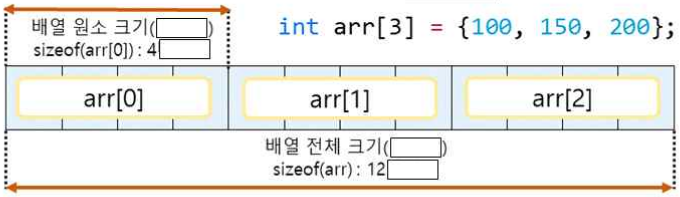
```
1  #include <stdio.h>
2  #define ROW 3
3  #define COL 3
4  int main(void) {
5      int arr[ROW][COL] = {{1,2,3},{4,5,6},{7, 8, 9}};
6      int *ptr = arr;
7      printf("ARRAY : %14s\t%14s\t%14s\n", "arr", "arr[0]", "arr[0][0]");
8      printf("Size : %14lu\t%14lu\t%14lu\n", sizeof(arr), sizeof(arr[0]), sizeof(arr[0][0]));
9      printf("Address : %p\t%p\t%p\n", arr, arr[0], &arr[0][0]);
10     printf("POINTER : %14s\t%14s\n", "dptr", "ptr");
11     printf("ADDRESS : %14s\t%14s\t%14s\n", "arr[0]", "arr[1]", "arr[2]");
12     printf("Array : %p\t%p\t%p\n", arr[0], arr[1], arr[2]);
13     printf("Pointer : %p\t%p\t%p\n", ptr+0*COL, ptr+1*COL, ptr+2*COL);
14     printf("VALUE : %14s\t%14s\t%14s\n", "arr[0][0]", "arr[1][1]", "arr[2][2]");
15     printf("Array : %14d\t%14d\t%14d\n", arr[0][0], arr[1][1], arr[2][2]);
16     printf("Pointer : %14d\t%14d\t%14d\n", *(ptr+0*COL+0), *(ptr+1*COL+1), *(ptr+2*COL+2));
17     return 0;
18 }
```

출력 예

```
main.c:6:13: warning: initialization from incompatible pointer type [-Wincompatible-pointer-types]
ARRAY   :           arr            arr[0]       arr[0][0]
Size    :            36               12              4
Address : 0x7ffefac30990 0x7ffefac30990 0x7ffefac30990
POINTER :          dptr              ptr
ADDRESS :        arr[0]           arr[1]         arr[2]
Array   : 0x7ffefac30990 0x7ffefac3099c 0x7ffefac309a8
Pointer : 0x7ffefac30990 0x7ffefac3099c 0x7ffefac309a8
VALUE   :     arr[0][0]        arr[1][1]      arr[2][2]
Array   :             1                5              9
Pointer :             1                5              9
```

7.4 포인터와 배열

- 배열 크기 계산
  - ✓ sizeof 연산자를 이용하여 배열 원소 수에 따라 크기를 계산 가능
    - ➢ sizeof 연산자는 저장공간의 크기를 [ ] 단위 반환
  - ✓ 1차원 배열은 sizeof(배열 이름)/sizeof(배열 원소) 계산
    - ➢ sizeof(배열 이름) : 배열 전체 공간의 [ ] 크기
    - ➢ sizeof(배열 원소) : 배열 각 원소의 [ ] 크기



7.4 포인터와 배열

- 배열 크기 계산
  - ✓ 2차원 배열 arr[3][3]의 [ ]는 sizeof(arr) / sizeof(arr[0])
    - ➢ sizeof(arr[0])는 행 하나 크기이므로, 전체 크기 / 행 크기 = [ ]
  - ✓ [ ]는 sizeof(arr[0]) / sizeof(arr[0][0]) 으로 구함
    - ➢ 행은 열의 모임이므로 열 하나의 크기 / 원소 크기 = [ ]

7.4 포인터와 배열

▪ 포인터 배열과 배열 포인터

✓ 배열은 여러 개의 동일한 데이터 타입을 저장하기 위한 자료구조

➢ 기본 데이터 타입 외에 포인터 저장 또한 가능

➢ 포인터를 원소로 가지는 배열을 포인터 배열(array of pointer)

✓ 포인터 배열의 선언 형식과 예시는 아래와 같음

자료형 *배열이름[배열크기];	int a, b, c; int *parr1[3] = {&a, &b, &c}; char *parr2[2]; double *parr3[6] = {NULL};

7.4 포인터와 배열

소스 7-15

```
#include <stdio.h>
int main(void) {
    float *arr[4];
    float f1 =114.1, f2 =200.5, f3 =31.0, f4 =5.123;
    arr[0] =&f1;
    arr[1] =&f2;
    arr[2] =&f3;
    arr[3] =&f4;
    printf("f1 : %.3f, *arr[0] : %.3f \n", f1, *arr[0]);
    printf("f2 : %.3f, *arr[1] : %.3f \n", f2, *arr[1]);
    printf("f3 : %.3f, *arr[2] : %.3f \n", f3, *arr[2]);
    printf("f4 : %.3f, *arr[3] : %.3f \n", f4, *arr[3]);
    printf("&f1 : %p, arr[0] : %p \n", &f1, arr[0]);
    printf("&f2 : %p, arr[1] : %p \n", &f2, arr[1]);
    printf("&f3 : %p, arr[2] : %p \n", &f3, arr[2]);
    printf("&f4 : %p, arr[3] : %p \n", &f4, arr[3]);
    return 0;
}
```

출력 예

```
f1 : 114.100, *arr[0] : 114.100
f2 : 200.500, *arr[1] : 200.500
f3 : 31.000, *arr[2] : 31.000
f4 : 5.123, *arr[3] : 5.123
&f1 : 0x7fff5480af5c, arr[0] : 0x7fff5480af5c
&f2 : 0x7fff5480af58, arr[1] : 0x7fff5480af58
&f3 : 0x7fff5480af54, arr[2] : 0x7fff5480af54
&f4 : 0x7fff5480af50, arr[3] : 0x7fff5480af50
```

7.4 포인터와 배열

- 포인터 배열과 배열 포인터

  ✓ 배열 포인터(pointer to array)는 [ ]을 가리키는 포인터

  ➢ 1차원 배열 `int arr[]`의 주소는 `int`형 포인터 변수에 저장 가능

  ➢ 2차원 배열 `arr[][3]`의 주소는 `int (*포인터 변수 명)[열 크기];` 형태로 구성된 포인터에 저장 Ex) `int (*ptr)[3];`

  ✓ 높은 우선순위의 후위 연산자 [] 때문에, 괄호를 생략하면 다른 의미로 변경

데이터 자료형 *포인터변수명; 포인터변수명 = 배열명;	데이터자료형 (*포인터변수명)[배열의 열크기]; 포인터변수명 = 배열명;
데이터 자료형 *포인터변수명 = 배열명;	데이터자료형 (*포인터변수명)[배열의 열크기] = 배열명;
`int arr[] = {1,2,3,4,5};` `int *ptr = arr;`	`int arr[2][3] = {{1,2,3},{4,5,6}}` `int (*ptr)[3] = arr;`

7.4 포인터와 배열

소스 7-16

```
1  #include <stdio.h>
2  int main(void) {
3      int arr1[3][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12}};
4      int arr2[7][4], arr3[1][10] = {0};
5      int(*parr)[4];
6      int i, j;
7      parr = arr1; // works
8      parr = arr2; // works
9      parr = arr3; // warning
10     for(i=0; i<10; i++)
11         printf("%d\t", (*parr)[i]);
12     printf("\n");
13     parr = arr1;
14     for(i=0; i<3; i++) {
15         for(j=0; j<4; j++)
16         printf("%d\t", parr[i][j]);
17         printf("\n");
18     }
19     return 0;
20 }
```

출력 예

```
main.c:9:7: warning: assignment from incompatible pointer type [-Wincompatible-pointer-types]
0       0       0       0       0       0       0       0       0       0
1       2       3       4
5       6       7       8
9       10      11      12
```

7.5 const와 포인터

- const와 포인터

  ✓ 포인터 변수를 선언할 때 const의 [ ]에 따라 다르게 동작

  ➢ [ ] 앞에 const 키워드를 사용하는 경우

  • 포인터 변수가 가리키는 값을 변경할 수 없는 상수로 지정하는 포인터

  ➢ [ ]과 [ ] 사이에 const 키워드를 사용하는 경우

  • 포인터 변수가 저장하고 있는 주솟값을 변경할 수 없는 포인터

```
int n1 = 10, n2 = 20;
const int *ptr1 = &n1;
int* const ptr2 = &n1;
int const *ptr3 = &n2;
```


7.6 Q&A

Q 23. char *p1, p2;처럼 선언을 하고 p2를 포인터로 사용하려는데 에러가 발생합니다. 이유는 무엇인가?

A. 다음 선언에서 잘못된 것은 없다. 대신 p1, p2 모두 포인터로 사용하려고 한다면 문제가 생긴다. Pointer 선언에서 *는 선언할 이름으로 구성된 declarator(선언자)의 일부분이다. 첫 번째 declarator는 "*p1"이며, *를 포함하고 있기 때문에, p1은 'a pointer to char', 즉 char를 가리키는 pointer가 된다. 하지만 p2의 declarator는 p2 이외에 다른 것을 가지고 있지 않으므로, 일반 char type의 변수 p2가 된다. 한 선언에서 두 개의 pointer를 선언하려면, char *p1, *p2;처럼 해야 한다.

7.6 Q&A

Q 24. 소스 파일에 char a[6]이라고 정의하고 extern char *a 라고 선언해 두었는데 왜 동작하지 않을까?

A. 소스 파일에 정의한 것은 문자(char)로 이루어진 배열이다. 그리고 선언한 것은 문자를 가리키는 포인터이다. 따라서 선언과 정의가 일치하지 않는 경우가 된다. 일반적으로, T 타입을 가리키는 포인터(pointer to type T)의 타입은 T 타입의 배열(array of type T)과 다르다. 대신 extern char a[]을 사용하는 것을 추천한다.

7.6 Q&A

Q 25. Q "포인터와 배열은 같다(equivalent)"라는 말은 어디에서 나온 것일까? (1/2)

A. C 언어에서 가장 혼동을 가져오는 부분은 바로 이 문장을 잘못 이해한 것에서 비롯한다. 배열과 포인터가 같다(equivalent)는 의미는 서로 같다는(identical) 말도 아니며, 서로 마음대로 바꿔쓸 수 있다(interchangeable)는 뜻도 아니다.
이 말이 의미하는 바는 다음과 같은 정의를 의미한다 : T 타입의 배열의 `l-value'가 수식에서 나타날 때, 배열의 첫 번째 요소를 가리키는 포인터로 변경(원문에는 decay, "퇴화하다"라고 표현되었음)된다. 변경된 포인터의 타입은 T타입을 가리키는 포인터가 된다. 여기에는 세 가지의 예외가 있다.

- 배열이 sizeof의 피연산자로 쓰일 때
- 배열이 & 연산자의 피연산자로 쓰일 때
- 문자 배열에서, 초기 값인 문자열로 쓰일 때

7.6 Q&A

Q 25. Q "포인터와 배열은 같다(equivalent)"라는 말은 어디에서 나온 것일까? (2/2)

A. 이 정의에 따라서, 내부적으로 배열이 포인터와는 매우 다르지만, 배열 혹은 포인터에 상관없이 [] 연산자를 쓸 수 있다. 배열 a와 포인터 p가 있다고 가정하고, a[i]는 위 정의에 따라서, 배열이 포인터로 변경되고, 이는 포인터 변수를 쓴 p[i]와 의미가 같아진다. 배열의 주소를 다음과 같이 포인터에 대입하면

```
p = a;
```

p[3]은 a[3]과 같은 곳을 가리키게 된다. 그렇기 때문에, 포인터를 사용하여 배열에 접근 이 가능하며, 함수 파라미터로 쓰일 수 있다.

7.6 Q&A

Q 26. 배열 안에 몇 개의 요소가 있는지 알고 싶으면 어떻게 해야 할까? sizeof 연산자는 배열의 크기를 byte 단위로 반환한다.

A. 소스 A 전체 배열의 크기를, 한 요소의 크기로 나누면 된다.

```
int array[] = { 1, 2, 3 };
int narray = sizeof(array) / sizeof(array[0]);
```

7.7 실습 및 과제

- 실습 및 과제 진행
  - DCU Code : http://code.cu.ac.kr
  - 7 주차 실습
    - 코딩 7-1, 코딩 7-2, 코딩 7-3, 코딩 7-4 (p267-293)
  - 7 주차 과제
    - 7장 프로그래밍 연습 1 ~ 2번 (p302)

7.8 참고문헌

- 메모리 주소 : https://ko.wikipedia.org/wiki/메모리_주소
- 포인터 : https://ko.wikipedia.org/wiki/포인터_(프로그래밍)

E N D