포인터 기초

01 포인터 변수와 선언

02 간접연산자 *와 포인터 연산

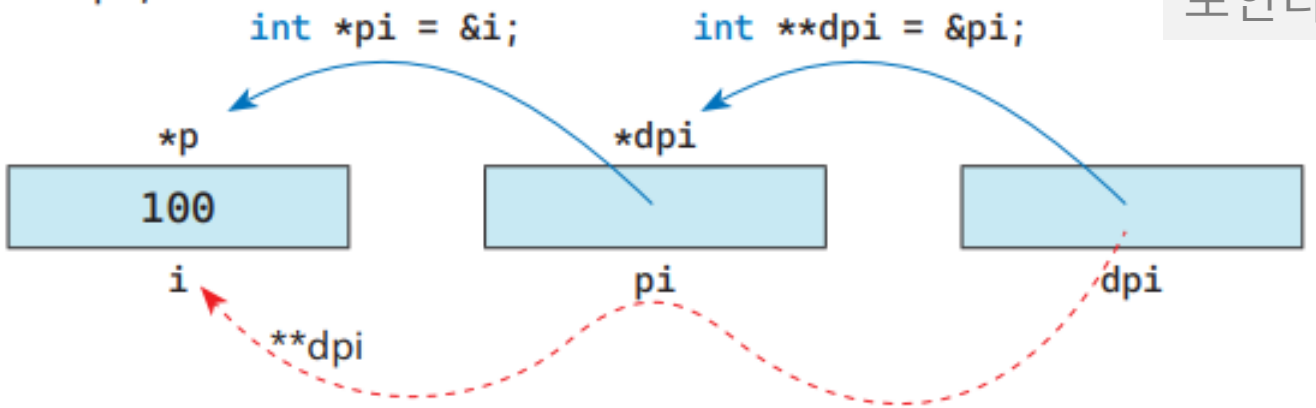
03 포인터 형변환과 다중 포인터

04 포인터를 사용한 배열 활용

이중 포인터

- 포인터 변수의 주소 값을 갖는 변수

```
int i = 20;
int *pi = &i;
int **dpi = &pi;
```




```
*pi = i + 2;        // i = i + 2;
**dpi = *pi + 2;    // i = i + 2;
```

그림 11-15 이중 포인터의 메모리와 변수

이중 포인터 활용
포인터를 함수 인자로 전달
다차원 배열 동적으로 할당
포인터 배열

- 삼중 포인터 : ???

실습예제 11-7 | Prj07 | 07multptr.c | 이중 포인터를 이용한 변수의 참조 | 난이도: ★

```c
#include <stdio.h>

int main(void)
{
   int i = 100;
   int *pi = &i;    // 포인터 선언
   int **dpi = &pi;// 이중 포인터 선언

   printf("%p %p %p\n", &i, pi, *dpi);

   *pi = i + 30;    // i = i + 30;
   printf("%d %d %d\n", i, *pi, **dpi);

   **dpi = *pi + 30;// i = i + 30;
   printf("%d %d %d\n", i, *pi, **dpi);

   return 0;
}
```

모두 변수 i의 주소값을 참조하는 방식

모두 변수 i 자체를 참조하는 방식

간접연산자와 증감연산자 활용

```c
int a[] = {10, 20};
int *p = &a[0];

printf("%d\n", *p++); // *(p++) 동일
printf("%d\n", *p);

p = &a[0];
printf("%d\n", *++p); // *(++p) 동일
printf("%d\n\n",*p);

p = &a[0];
printf("%d\n", (*p)++);
printf("%d\n", *p);

a[0] = 10;
p = &a[0];
printf("%d\n", ++*p); // ++(*p) 동일
printf("%d\n\n", *p);
```

포인터와 증감 연산자 활용
배열 각 원소 접근
동적 할당 메모리 순회
문자열 포인터 처리/순회

포인터 상수

```
int i = 10, j = 20;
```

❶ `const int *p = &i;`

```
*p = 20; //오류 발생
```

(지역 변수) const int *p
*p가 상수로 *p로 수정할 수 없음
온라인 검색

❷ `int const *p = &i;`

```
*p = 20; //오류 발생
```

(지역 변수) const int *p
*p가 상수로 *p로 수정할 수 없음
온라인 검색

❸ `int* const p = &i;`

```
p = &j; //오류 발생
```

포인터 변수 자체가 상수

```c
#include <stdio.h>

// 배열의 요소를 출력하는 함수 선언
void print(const int* ptr, int size);

int main(void) {
    // 상수 배열 선언 및 초기화
    const int arr[] = { 1, 2, 3, 4, 5 };

    // 배열을 출력하는 함수 호출
    print(arr, 5);

    return 0;
}

void print(const int* ptr, int size) {
    // 배열 요소를 순회하면서 출력
    for (int i = 0; i < size; i++) {
        printf("%d ", ptr[i]);
    }
    printf("\n");
}
```

```
16  void print(const int* ptr, int size) {
17      // 배열 요소를 순회하면서 출력
18      for (int i = 0; i < size; i++) {
19          printf("%d ", ptr[i]);
20          ptr[i] = 5; // 이 줄은 상수 배열에 대한 잘못된 접근입니다.
21      }
22      printf("\n");
23  }
```

포인터 기초

01 포인터 변수와 선언

02 간접연산자 *와 포인터 연산

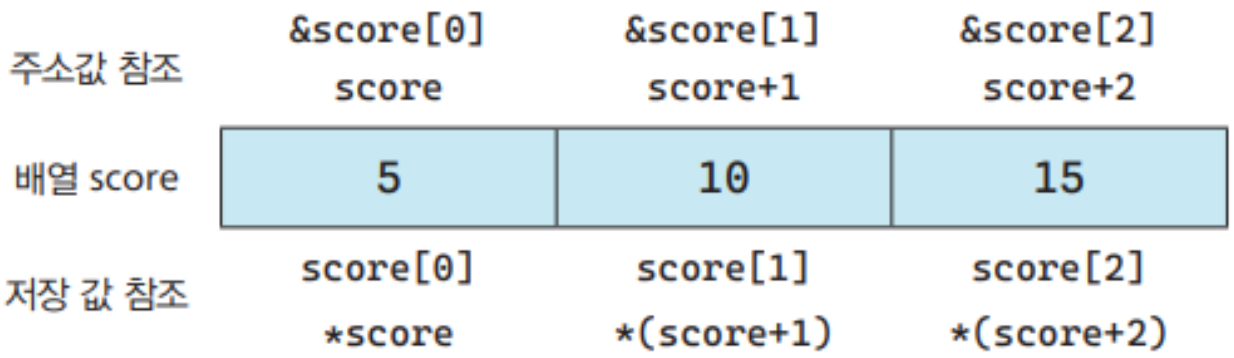
03 포인터 형변환과 다중 포인터

04 포인터를 사용한 배열 활용

배열이름을 이용한 참조

```
int score[] = {5, 10, 15};
```

배열 이름
배열의 첫 번째 원소를 가리키는 포인터



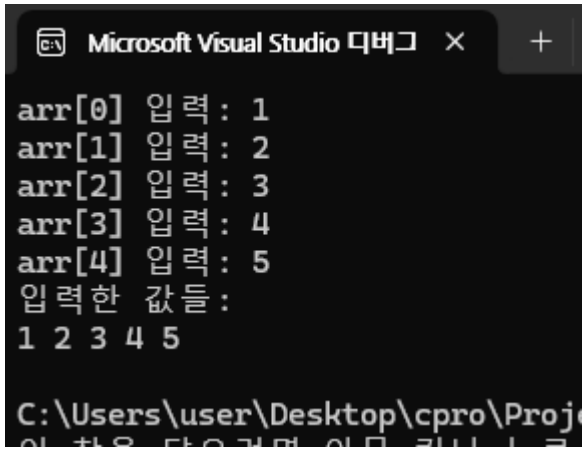
- 배열 (i+1)번째 원소 주소값: (score + i), &score[i]
- 배열 (i+1)번째 원소 저장 값: *(score + i), score[i]

원소의 주소와 다양한 접근 방법

배열 초기화 문장		int score[] = {10, 20, 30};		
원소 값		10	20	30
배열원소 접근 방법	score[i]	score[0]	score[1]	score[2]
	*(score+i)	*score	*(score+1)	*(score+2)
주소값(첫 주소 + 배열원소 크기*i)		100	104 (100 + 1*4)	108 (100 + 2*4)
주소값 접근 방법	&score[i]	&score[0]	&score[1]	&score[2]
	score+i	score	score+1	score+2

정수형 배열 5개를 선언하고, 포인터를 이용하여 다음을 수행하시오:

1. 배열의 모든 값을 입력받아 저장
2. 포인터로 각 원소에 접근하여 출력




```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

#define SIZE 5

void inputArray(int *arr, int size);
void printArray(const int *arr, int size);

int main(void) {
    int arr[SIZE];

    inputArray(arr, SIZE);
    printArray(arr, SIZE);

    return 0;
}
```

원소의 주소와 다양한 접근 방법

```
int a[3] = {5, 10, 15};
int* p = a; //a == &a[0]

//포인터 변수 p 사용, 배열 원소 값 참조
printf("%d %d %d\n", *(p), *(p + 1), *(p + 2));
//위 포인터 변수 p에서 배열처럼 첨자를 사용 가능
printf("%d %d %d\n", p[0], p[1], p[2]);
```

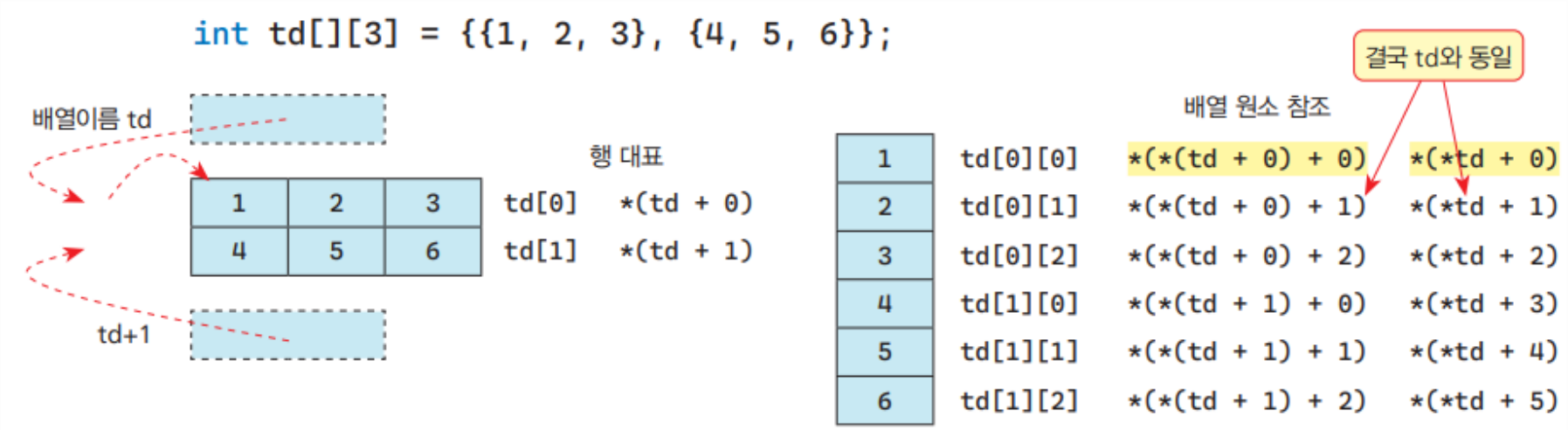
포인터 변수와 증감연산자 활용

```c
int a[3] = {5, 10, 15};

int* p = a; //a == &a[0]

//a[0]을 출력 후, p 다음 주소로 증가
printf("%d ", *p++);
//a[1]을 출력
printf("%d\n\n", *p);
```

```c
int a[3] = {5, 10, 15};
int *p = &a[2];

printf("%d ", *p--);
printf("%d\n", (*p)--);

printf("%d %d %d\n", *(p - 1), *p, *(p + 1));
printf("%d %d %d\n", p[-1], p[0], p[1]);
```

배열이름과 간접연산자로 참조

- 이차원 배열, 배열이름인 td는 ...
- td[i]는 ...



- *(p[i] + j)
  - (i+1) 행, (j+1)열 원소 참조
- *(*(p + i) + j)
  - (i+1) 행, (j+1)열 원소 참조

```c
int td[2][3] = { { 1, 2, 3 }, { 4, 5, 6 } };

// 배열 및 행 크기 출력
printf("%u\n", (unsigned int)sizeof(td));     // 배열 전체 크기
printf("%u\n", (unsigned int)sizeof(td[0]));  // 첫 번째 행 크기

// 값 수정
*(*td + 4) = 20;       // td[1][1] = 20
*(*(td + 1) + 2) = 30; // td[1][2] = 30

// 수정된 값 출력
printf("%u\n", td[0][0]); // td[0][0]
printf("%u\n", td[1][1]); // td[1][1]
printf("%u\n", td[1][2]); // td[1][2]
```

2차원 배열 포인터 선언

```
원소자료형 *변수이름;
변수이름 = 배열이름;
또는
원소자료형 *변수이름 = 배열이름;
```

```
원소자료형 (*변수이름)[ 배열열크기];
변수이름 = 배열이름;
또는
원소자료형 (*변수이름)[ 배열열크기] = 배열이름;
```

```
int a[] = {8, 2, 8, 1, 3};
int *p = a;
```

```
int ary[][4] = {5, 7, 6, 2, 7, 8, 1, 3};
int (*p)[4] = ary;   //열이 4인 배열을 가리키는 포인터
//int *p[4] = ary;   //포인터 배열
```

int arr[2][3][4];

→ 3차원 배열 포인터 p 선언 : ....

```c
#include <stdio.h>
#define ROWS 2
#define COLS 3

void doubleElements(int arr[ROWS][COLS]);
void printArray(int arr[ROWS][COLS]);

int main() {
    int arr[ROWS][COLS] = {{1, 2, 3}, {4, 5, 6}};

    printf("Original array:\n");
    printArray(arr);

    doubleElements(arr);

    printf("Array after doubling elements:\n");
    printArray(arr);

    return 0;
}
```

```c
void doubleElements(int arr[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            arr[i][j] *= 2;
        }
    }
}

void printArray(int arr[ROWS][COLS]) {
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
}
```

```c
#include <stdio.h>
#define COLS 3
// 이차원 배열의 각 원소를 두 배로 만드는 함수
void doubleElements(int (*ptr)[COLS], int rows);

int main() {

    int arr[2][COLS] = {{1, 2, 3}, {4, 5, 6}};

    doubleElements(arr, 2);

    printf("Array after doubling elements:\n");
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < COLS; j++) {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

```c
void doubleElements(int (*ptr)[COLS], int rows) {
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < COLS; j++) {
            ptr[i][j] *= 2;
        }
    }
}
```

포인터 배열 개요와 선언

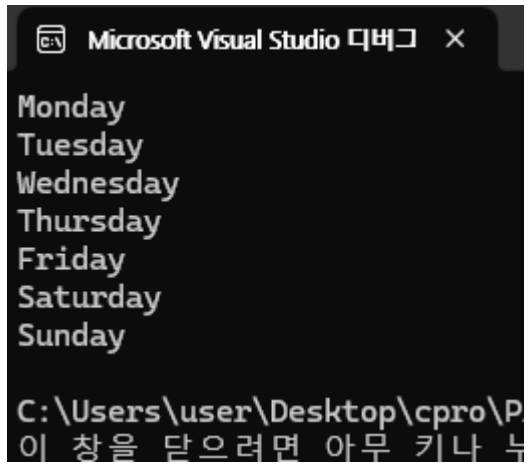
포인터 배열 변수 선언

자료형 *변수이름[배열크기] ;

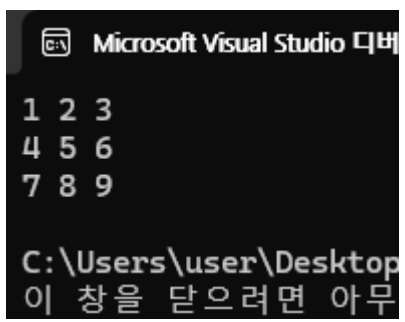
```
int *pary[5];
char *ptr[4];
float a, b, c;
double *dary[5] = {NULL};
float *ptr[3] = {&a, &b, &c};
```

포인터 배열 활용
문자열
2차원 동적 배열

```c
#include <stdio.h>

int main() {

    const char *days[7] = {
        "Monday", "Tuesday", "Wednesday",
        "Thursday", "Friday", "Saturday", "Sunday"
    };


    for (int i = 0; i < 7; i++) {
        printf("%s\n", days[i]);
    }

    return 0;
}
```

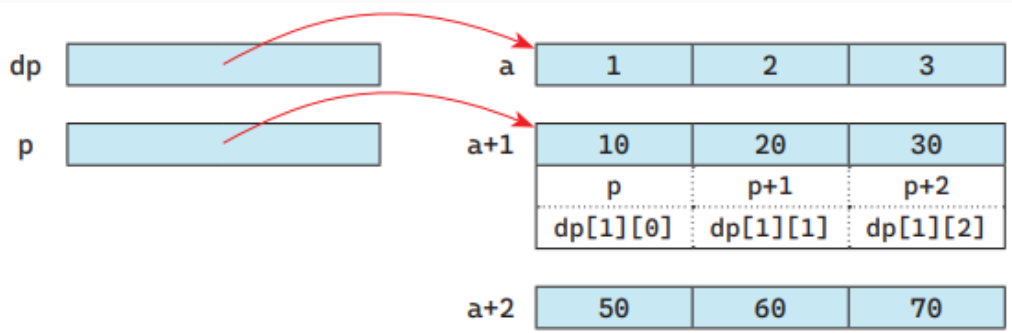


```c
#include <stdio.h>

int main() {

    int arr1[] = {1, 2, 3};
    int arr2[] = {4, 5, 6};
    int arr3[] = {7, 8, 9};

    int *arrPointers[3] = {arr1, arr2, arr3};

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            printf("%d ", arrPointers[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```



lab4ptrs.c 난이도: ★★

```c
#include <stdio.h>

int main(void)
{
    int a[][3] = { {1, 2, 3}, {10, 20, 30}, {50, 60, 70} };
    int (*dp)[3] = a;
    int* p = a[1];

    printf("%d %d %d\n", *p, *(p + 1), *(p + 2));
    printf("%d %d %d\n", dp[1][0], dp[1][1], dp[1][2]);

    return 0;
}
```



포인터 기초

- **포인터 변수** 이해
  - 메모리와 주소, 주소연산자 &
  - ***를 사용한** 포인터 변수 선언과 **간접참조** 방법
  - **포인터 변수의 연산과 형변환**
- **다중 포인터**와 **배열 포인터**를 이해
  - 이중 포인터의 필요성과 선언 및 사용 방법
  - 증감연산자와 포인터와의 표현식
  - 포인터 상수
- **배열과 포인터 관계** 이해
  - **배열이름은 포인터 상수**이며 포인터 변수로도 참조
  - 1차원과 2차원 배열의 배열 포인터 활용
  - **포인터 배열**