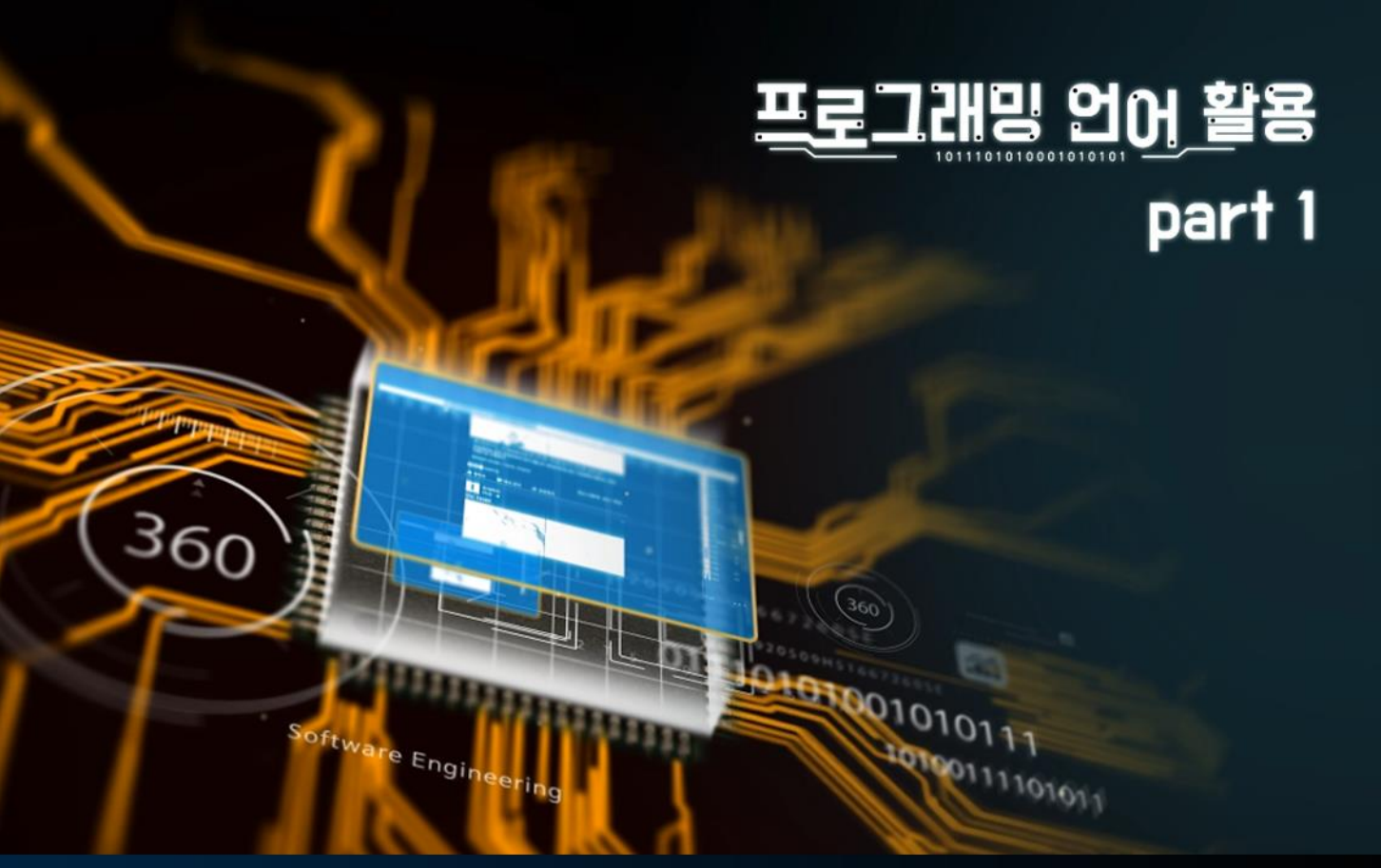


포인터 활용

한국기술교육대학교
온라인평생교육원
OLEI

학습내용

- 포인터와 배열
- 포인터 연산

학습목표

- 포인터와 배열의 관계에 대해 설명할 수 있다.
- 포인터를 이용한 연산을 사용할 수 있다.

포인터와 배열

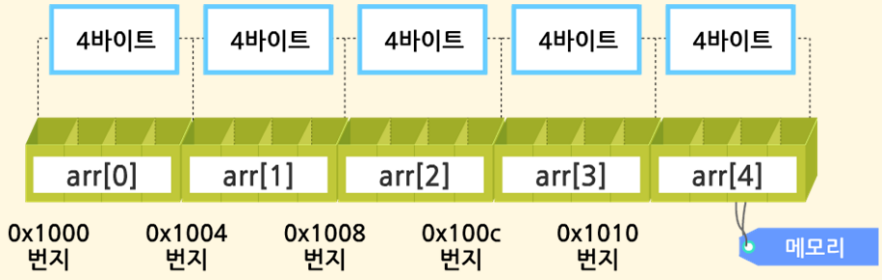
1 포인터로 배열 참조

배열명 배열의 시작 주소를 의미하는 상수

1 배열의 시작 주소를 구할 때는 & 없이 배열명만 사용

```
int arr[5];

arr == &arr[0]
```



배열의 시작 주소는
첫 번째 원소의 주소와 같음

2 배열명을 포인터처럼 사용 가능

- 인덱스를 사용하는 대신 **배열의 시작 주소로 포인터 연산**을 하면 배열의 특정 원소에 접근 가능

포인터와 배열

1 포인터로 배열 참조

3 *(arr+i)는 arr[i]를 의미

- 배열의 시작 주소에서 데이터 타입 i개 크기만큼 증가된 주소에 있는 값

```
*arr = arr[0]
```

```
&arr[i] = arr + i
```

배열의 i번째 원소의 주소

배열 시작 주소에서 i개의 원소만큼 떨어진 주소

```
arr[i] = *(arr + i)
```

배열의 i번째원소

배열 시작 주소에서 i개의 원소만큼 떨어진 주소에 있는 값

4 배열의 시작주소로 초기화된 포인터를 이용해서 배열의 모든 원소에 접근가능

5 포인터 변수를 배열 이름인 것처럼 사용 가능

```
int a[4];
int *p = a;
```

```
*(p + i) == p[i]
```

포인터가 가리키는 곳에서 i개의 원소만큼 떨어진 주소에 있는 값

포인터가 가리키는 배열의 i번째 원소

```
p + i == &p[i]
```

포인터가 가리키는 곳에서 i개의 원소만큼 떨어진 주소

포인터가 가리키는 배열의 i번째원소의 주소

포인터와 배열

1 포인터로 배열 참조

6 포인터와 +, - 연산

p + N

p가 가리키는 데이터형 N개 크기만큼 증가된 주소가 연산의 결과

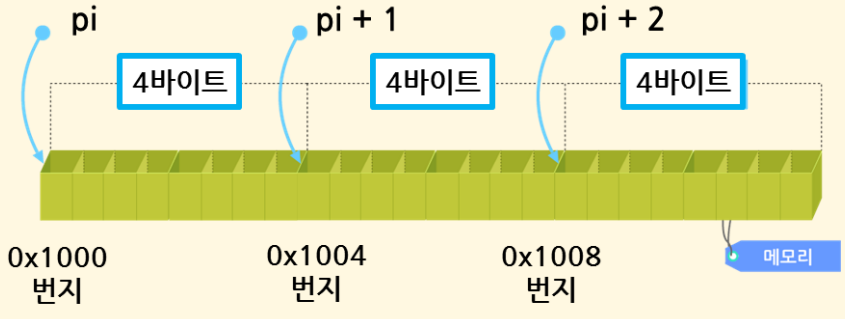
p - N

p가 가리키는 데이터형 N개 크기만큼 감소된 주소가 연산의 결과

```
int n;
int *pi = &n;
```

pi + i

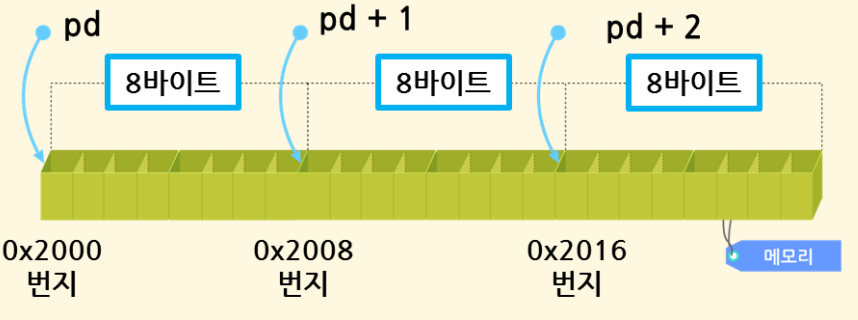
int i개 크기만큼 증가된 주소




```
double d;
double *pd = &d
```

pd + i

double i개 크기만큼 증가된 주소



포인터와 배열

1 포인터로 배열 참조

6 포인터와 +, - 연산

- '포인터 - 포인터' 연산은 두 포인터의 차를 구하는 데 사용

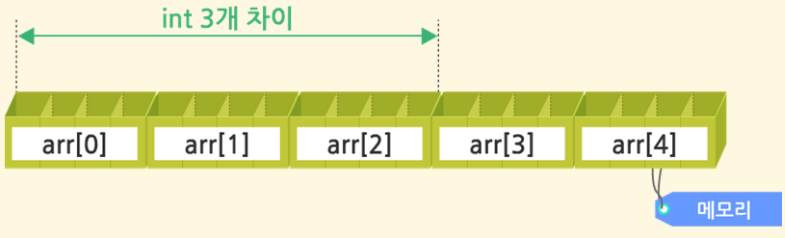
```
int arr[5]
int diff = &arr[3] - &arr[0]
```

arr[3]과 arr[0]은 int 3개만큼 떨어져 있음

```
int arr[5];
int diff = &arr[3] - &arr[0];
```

&arr[3] - &arr[0]

int 3개만큼 주소가 차이나므로 연산의 결과는 3이 됨




```
01:
02: #include <stdio.h>
03:
04: int main(void)
05: {
06:         char ch;
07:         char *pc = &ch;
08:
09:         int n;
10:         int *pi = &n;
```

char*형 변수의 선언

int*형 변수의 선언

포인터와 배열

1 포인터로 배열 참조

```
11:
12:         double d;
13:         double *pd = &d;
14:
15:         int arr[3];
16:         int i;
17:
18:         for(i = 0 ; i < 3 ; i++)
19:             printf("pc+%d = %p₩n", i, pc+i);
20:
21:         for(i = 0 ; i < 3 ; i++)
22:             printf("pi+%d = %p₩n", i, pi+i);
23:
24:         for(i = 0 ; i < 3 ; i++)
25:             printf("pd+%d = %p₩n", i, pd+i);
26:
27:         for(i = 0 ; i < 5 ; i++)
28:             printf("&arr[%d]-&arr[0] = %d₩n", i, &arr[i]-
&arr[0]);
29:
30:         return 0;
31: }
```

double*형 변수의 선언

char*형에 정수를 더하는 연산

int*형에 정수를 더하는 연산

double*형에 정수를 더하는 연산

포인터의 차를 구하는 연산

포인터와 배열

1 포인터로 배열 참조

```
int arr[5] = { 12, 25, 37, 49, 53 } ;     int *p=arr;
```

*p+1	13
*(p+2)	37
arr+3	arr[3]의 주소
*(arr+3)	49
*arr+4	16
p[4]	53
&arr[2]	arr[2]의 주소
p+3	arr[3]의 주소

```
#include <stdio.h>

const int MAX = 3;

int main ()
{

   int  var[] = {100, 200, 300};
   int i;
```

포인터와 배열

1 포인터로 배열 참조

```
    for (i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d₩n", i, *(var+i) );
    }

    return 0;
}
```

```
#include <stdio.h>

const int MAX = 3;

int main ()
{
      int  var[] = {100, 200, 300};
    int *parr = var;
      int i;
```

포인터와 배열

1 포인터로 배열 참조

```
    for (i = 0; i < MAX; i++) {
        printf("Pointer Value of var[%d] = %d₩n", i, *(parr+i) );
        printf("Index Value of var[%d] = %d₩n", i, parr[i] );
    }

    return 0;
}
```

2 포인터와 배열 원소

1 배열의 원소를 가리키는 포인터는 배열의 어떤 원소든지 가리킬 수 있음

```
int arr[5] = {10, 20, 30, 40, 50};
int *p = &arr[2];          ← p는 arr[2]를 가리킴
printf("p[0] = %d", p[0]);  ← p[0]은arr[2]를의미하므로30을출력함
printf("p[1] = %d", p[1]);  ← p[1]은arr[3]를의미하므로40을출력함
```

포인터와 배열

2 포인터와 배열 원소

2 포인터가 배열의 원소가 아닌 일반 변수를 가리킬 때도 *(p+i) == p[i]는 항상 성립함

```
short data = 10;

short *ptr = &data;

ptr[0] = 20;
```

*p 대신 p[0]처럼 사용할 수 있음

배열과 포인터의 차이점

- 배열이 메모리에 할당되고 나면, **배열의 시작 주소 변경 불가**
- 포인터 변수는 값을 변경할 수 있으므로, **포인터 변수에 보관된 주소는 변경 가능**

포인터와 배열

2 포인터와 배열 원소

```
int x[5];
int y[5];
```

컴파일 에러

```
x = y;
x++;
```

배열의 시작 주소는 변경할 수 없음

```
int x[5];
int y[5];
int *p = x;
```

```
p = y;
p++;
```

올바른 문장

포인터에 저장된 주소는 변경할 수 있음

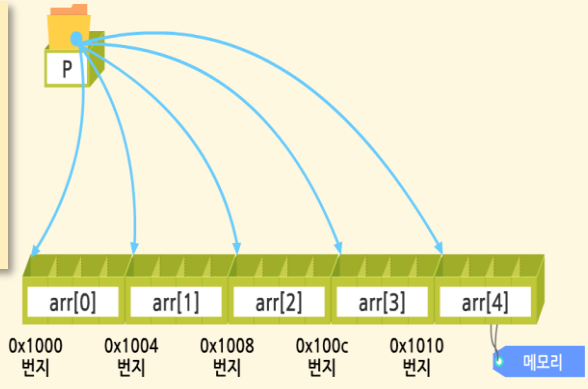
포인터 연산

1 포인터 증감 연산

포인터와 ++, -- 연산

포인터에 대한 증감 연산(++, --)도 포인터형에 의해 연산의 결과가 결정

```
int arr[5];
int *p = &arr[0];
for( i = 0 ; i <5 ; i++, p++ )
    printf("&d \ n", *p );
```



p++;

sizeof(int)만큼 주소 증가

```
#include <stdio.h>
int main(void)
{
        int arr[5] = {10, 20, 30, 40, 50};
        int *p = &arr[0];
        int i;
```

int *형 변수의 선언

포인터 연산

1 포인터 증감 연산

```
        for(i = 0 ; i < 5 ; i++, p++)
            printf("%d₩n", *p);
        return 0;
}
```

포인터 변수의 증가 연산

우선 순위 : p++ > ++p = *

```
*p++   =   *(p++)
++*p   =   ++(*p)
```

```
int arr[5] = {10, 20, 30, 40, 50};
int *p = arr;
```

```
*p++ ?
(*p)++ ?
```

포인터 연산

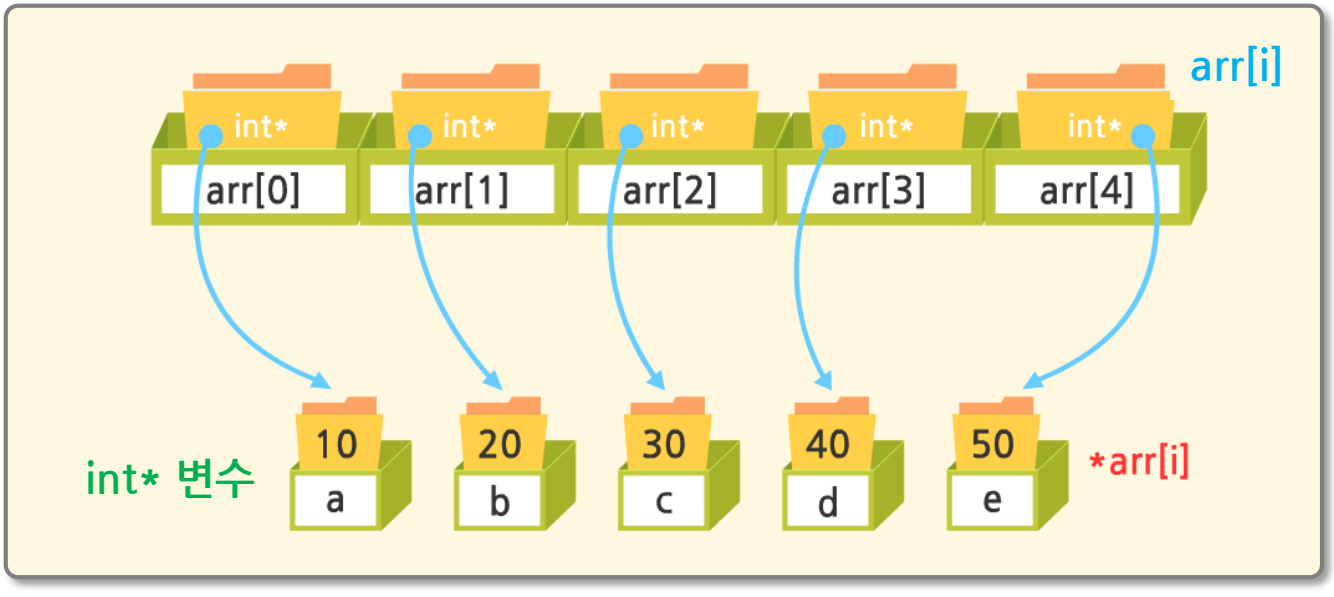
2 포인터 배열 처리

포인터 배열 주소를 저장하는 배열

형식 데이터 타입 *포인터 변수명[크기];

```
int a = 10, b = 20, c = 30, d = 40, e = 50;
int* arr[5] = {&a, &b, &c, &d, &e};
```

*arr[i]
int* 변수
int 변수



포인터 연산

2 포인터 배열 처리

```
int i;
int a=1, b=2, c=3, d=4, e=5;
int* arr[5] = {&a, &b, &c, &d, &e};
for( i = 0 ; i < 5 ; i++ )
    printf("%d ", *arr[i]);
```

1 포인터 배열의 각 원소에 배열의 시작 주소를 저장할 수도 있음

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

int y[3] = {4, 5, 6};

int z[3] = {7, 8, 9};

int* arr[3] = {x, y, z};
```

포인터 배열의 원소를 int 배열의 시작 주소로 초기화함

포인터 연산

2 포인터 배열 처리

2 arr[i]가 int 배열의 시작 주소로 초기화되었을 때, arr[i]가 가리키는 배열의 원소에 접근하려면 arr[i][j]로 기술

```
for( i = 0 ; i < 3 ; i++ )
{
    for( j = 0 ; j < 3 ; j++ )
        printf( "%d", arr[i][j] );  ——— *(arr[i]+j)와 같은 의미
    printf( "\n" );
}
```

학습정리

1. 포인터와 배열

- 배열명은 배열의 시작 주소를 의미함
- 포인터 변수를 배열명으로 초기화한 경우 포인터 변수를 배열처럼 인덱스를 사용하는 것이 가능함
- 배열명은 변수가 아니므로 증감 연산자에 의한 연산은 불가능함

2. 포인터 이용

- 배열명을 포인터 변수와 같이 연산에 의해 배열요소를 참조할 수 있음
- 포인터 변수에 증감 연산자를 이용하여 배열요소를 참조할 수 있음
- 후위 증감 연산자가, 전위 증감 연산자보다 우선순위가 높음