SEOIL UNIVERSITY

포인터

포인터


```
#include <iostream>
using namespace std;

int main()
{
	int num = 0;
	int* p;
	p = &num; //포인터 초기화
	num = 100;
//	*p = 50;
	cout << num << endl;
	cout << &num << endl;
	cout << p << endl;
	cout << *p << endl;
	cout << &p << endl;
//	cout << *num << endl;
	return 0;
}
```

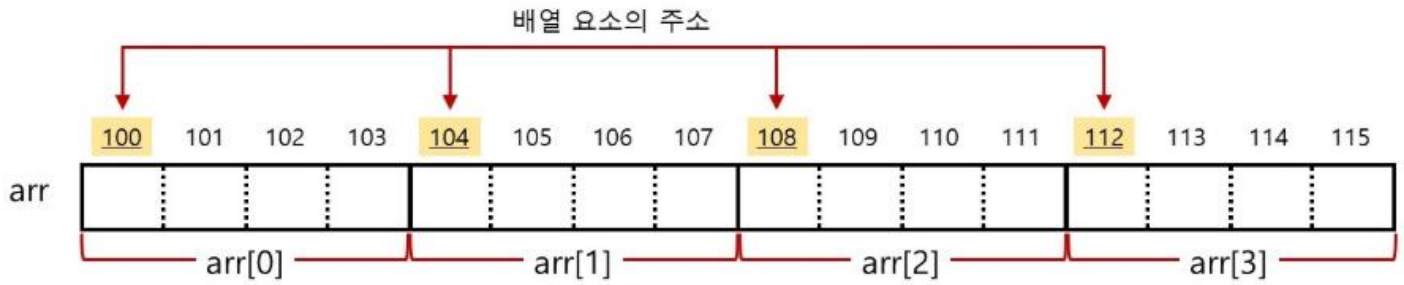
포인터와 배열의 관계


```
int main(void)
{
    int arr[3] = { 1, 2, 3 };

    printf("배열의 이름 : %p₩n", arr);

    printf("첫 번째 요소 : %p₩n", &arr[0]);
    printf("두 번째 요소 : %p₩n", &arr[1]);
    printf("세 번째 요소 : %p₩n", &arr[2]);

    return 0;
}
```



포인터 배열


```
int main() {
        int nums[5] = { 10, 20, 30, 40, 50 };
        int* p;

        p = nums;   // int* p = nums;

        cout << nums << endl;
        cout << &nums << endl;
        cout << p << endl;
        cout << *p << endl;
        cout << &p << endl;
        cout << *(nums)+1 << endl;
        cout << *(nums+1) << endl;
        cout << *p + 2 << endl;
        cout << *(p + 2) << endl;
        cout << p[3] << endl;
        return 0;
}
```

포인터 연산

포인터 연산은 +, -, ++, -- 연산자를 사용
포인터 값을 증가/감소시키는 연산
4바이트인 int형 변수의 주소 100번지에 1을 더한 결과는 101이 아닌 104
연산 결과 또한 주소

```
int main(void)
{
    char* ptr1 = 0;
    int* ptr2 = 0;
    double* ptr3 = 0;

    printf("%d번지, %d번지, %d번지₩n", ptr1, ptr2, ptr3);

    ptr1++;
    ptr2++;
    ptr3++;

    printf("%d번지, %d번지, %d번지₩n", ptr1, ptr2, ptr3);

    return 0;
}
```

포인터와 배열

배열의 이름과 포인터 변수는 상수, 변수의 차이만 있을 뿐, 사실은 동일
따라서 ptr에 저장된 값이 arr의 주소값

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

```
int main(void)
{
    int arr[3] = { 1, 2, 3 };

    int* ptr = arr;

    printf("%d %d %d₩n", *ptr, *(ptr + 1), *(ptr + 2));
    printf("%d %d %d₩n", ptr[0], ptr[1], ptr[2]);
    printf("%d %d %d₩n", *arr, *(arr + 1), *(arr + 2));
    printf("%d %d %d₩n", arr[0], arr[1], arr[2]);

    return 0;
}
```

배열 포인터

배열을 가리킬 수 있는 '포인터'

1차원 배열은 이름 자체가 포인터이기 때문에 배열 포인터가 필요없음
2차원 배열은 이름 자체로 포인터의 역할을 할 수 없음
배열 포인터를 사용하는 이유는 2차원 이상의 배열을 가리킬 때 포인터를 배열처럼 사용
따라서 배열 포인터는 2차원 이상의 배열에서만 의미가 있음

```
int main(void)
{
    int arr2d[2][3] = {
    {10, 20, 30},
    {40, 50, 60},
    };

    printf("%d %d₩n", *arr2d[0], *arr2d[1]);

    return 0;
}
```

배열 포인터

배열 포인터 정의

자료형 (*포인터이름)[가로(행)길이]

```
int (*ptr)[3]
```

int : 가리킬 수 있는 대상에 대한 정보(즉, int형 변수를 가리키는 포인터)
(*ptr) : ptr은 포인터임을 의미
[3] : 포인터 연산에 따른 증가/감소의 폭
(값을 1 증가시 int형이므로 4바이트*3 = 12바이트 이동)

배열 포인터


```
int main(void)
{
    int arr2d[2][3] = {
    {10, 20, 30},
    {40, 50, 60},
    };

    int i, j;

    int(*ptr)[3] = arr2d;    // 배열 포인터 선언

    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%d ", ptr[i][j]);    // 배열 포인터로 참조
        printf("₩n");
    }

    return 0;
}
```

포인터 배열, 배열 포인터

포인터 '배열'은 주소값들을 저장하는 '배열'
배열 '포인터'는 배열의 시작주소값을 저장할 수 있는 '포인터'

`int *ptr[3];` 첫 번째는 int형 변수로 이루어진 int형 포인터 배열

`int (*ptr)[3];` 두 번째는 가로길이가 3인 int형 2차원 배열을 가리키는 포인터 변수

( ) (괄호)의 유무가 중요

Call By Value


```
void add(int n1, int n2) {
        int sum = n1 + n2;
        cout <<  “sum = " << sum << endl;
}

int main()
{
        int num1 = 100, num2 = 200;
        add(num1, num2);
        cout << num1 << " : " << num2 << endl;
        return 0;
    }
```

Call By Value

함수 호출 시 인수로 전달되는 변수의 참조 값을 함수 내부로 전달하는 방식

```
int swap(int n1, int n2) {
	int temp;
	temp = n1;
	n1 = n2;
	n2 = temp;
}

int main()
{
	int num1 = 100, num2 = 200;
	cout << num1 << " : " << num2 << endl;
	swap(num1, num2);
	cout << num1 << " : " << num2 << endl;
	Return 0;
}
```

Call by Reference


```
void add(int* n1, int* n2) {
        int sum = *n1 + *n2;
        cout <<  “sum = " << sum << endl;
}

int main()
{
        int num1 = 100, num2 = 200;
        add(&num1, &num2);
        cout << num1 << " : " << num2 << endl;
        return 0;
    }
```

Call by Reference

함수 호출 시 인수로 전달되는 변수의 참조 값을 함수 내부로 전달하는 방식

```
void swap(int *n1, int *n2) {

cout << *n1 << " : " << *n2 << endl;
	int temp;
	temp = *n1;
	*n1 = *n2;
	*n2 = temp;
	cout << *n1 << " : " << *n2 << endl;
}

int main()
{
	int num1 = 100, num2 = 200;
	cout << num1 << " : " << num2 << endl;
	swap(&num1, &num2);
	cout << num1 << " : " << num2 << endl;
	return 0;
}
```

참조자


```
int change_val(int* p) {
	*p = 3;
	return 0;
}

int main() {
	int number = 5;
	cout << number << endl;
	change_val(&number);
	cout << number << endl;
	return 0;
}
```

참조자


```
int main() {
	int a = 3;
	int& another_a = a;
	another_a = 5;
	cout << "a : " << a << endl;
	cout << "another_a : " << another_a << endl;
	cout << "a : " << &a << endl;
	cout << "another_a : " << &another_a << endl;
	return 0;
}
```

참조자

- **레퍼런스는 정의 시에 반드시 누구의 별명인지 명시 해야**
- **불가 : int**& another_a;
- **가능 : int*** p;
- **레퍼런스가 한 번 별명이 되면 절대로 다른 이의 별명이 될 수 없음**

참조자


```
int main() {
	int a = 10;
	int& another_a = a; // another_a 는 a 의 참조자!
	int b = 3;
	another_a = b; // ??

	cout << another_a << endl;

	cout << a<< endl;
	cout << b<< endl;
	return 0;
}
```

a = b 와 동치

참조자


```
int main() {
	int a = 10;
	int b = 5;

	int* p = &a; // p 는 a 를 가리킨다.

	cout << *p << endl;

	p = &b; // p 는 a 를 버리고 b 를 가리킨다

	cout << *p << endl;
	return 0;
}
```

포인터는 다른 주소를 가질 수 있음

참조자

- 포인터의 경우
- 포인터 p 를 정의 한다면
  - int a = 10;
- int* p = &a; // p 는 메모리 상에 메모리를 차지

- 레퍼런스의 경우
- int a = 10;
- int &another_a = a; //another_a 메모리를 차지할 필요가 있을까?

참조자

함수 인자로 레퍼런스 받기

```
int change_val(int& p) {
    p = 3;
    return 0;
}
int main() {
    int number = 5;
    cout << number << endl;
    change_val(number);
    cout << number << endl;
}
```

p 가 정의되는 순간은 change_val(number) 로 호출할 때 이므로
int& p = number 가 실행

참조자


```
int main() {
	int x;
	int& y = x;
	int& z = y;
	x = 1;
	cout << "x : " << x << " y : " << y << " z : " << z << endl;
	y = 2;
	cout << "x : " << x << " y : " << y << " z : " << z << endl;
	z = 3;
	cout << "x : " << x << " y : " << y << " z : " << z << endl;
}
```

실제로 C++ 문법 상 참조자의 참조자를 만드는 것은 금지
y 와 z 모두 x 의 참조자가 된것

레퍼런스 배열

SEOIL UNIVERSITY

```
int a, b;
int& arr[2] = {a, b};
```

에러 발생
레퍼런스의 레퍼런스, 레퍼런스의 배열, 레퍼런스의 포인터는 존재할 수 없다.

```
int main() {
int arr[3] = { 1, 2, 3 };
int(&ref)[3] = arr;
ref[0] = 2;
ref[1] = 3;
ref[2] = 1;
cout << arr[0] << arr[1] << arr[2] << endl;
return 0;
}
```

ref 가 arr 를 참조

레퍼런스 리턴 함수


```
int function() {
	int a = 2;
	return a;
}

int main() {
	int b = function();
	cout << b << endl;
	return 0;
}
```

function 안에 정의된 a 라는 변수의 값이 b 에 **복사**
function 이 종료되고 나면 a 는 메모리에서 삭제

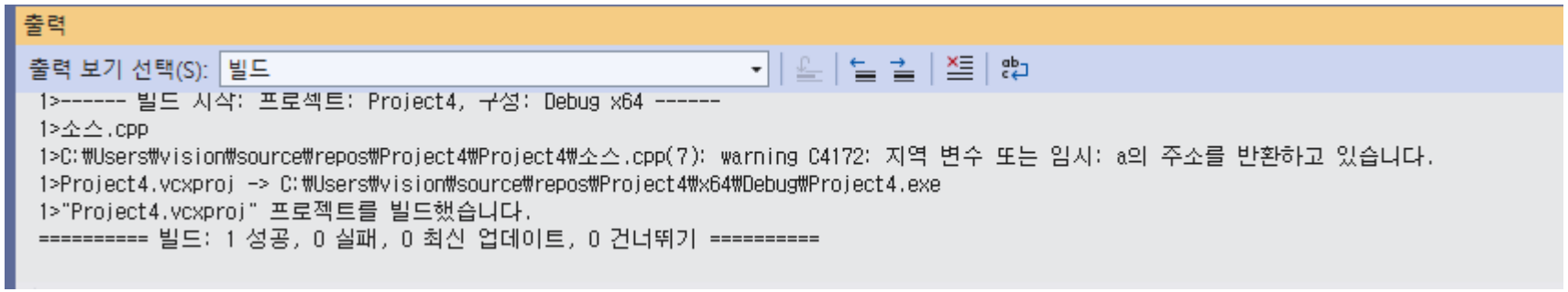
지역변수의 레퍼런스를 리턴?

SEOIL UNIVERSITY

```
int &function() {
	int a = 2;
	return a;
}

int main() {
	int b = function();
	cout << b << endl;
	return 0;
}
```

런타임 에러



외부 변수의 레퍼런스를 리턴


```
int& function(int& a) {
	a = 5;
	return a;
}
int main() {
	int b = 2;
	int c = function(b);
	cout << b << endl;
	cout << c << endl;
	return 0;
}
```

인자로 받은 레퍼런스를 그대로 리턴

function(b) 를 실행한 시점에서 a 는 main 의 b 를 참조

function이 리턴한 참조자는 **아직 살아있는 변수인 b 를 계속 참조**

포인터


```
int main() {
        int* p = new int;
        *p = 10;
        cout << *p << endl;
        delete p;
        return 0;
}
```

int 영역이 할당

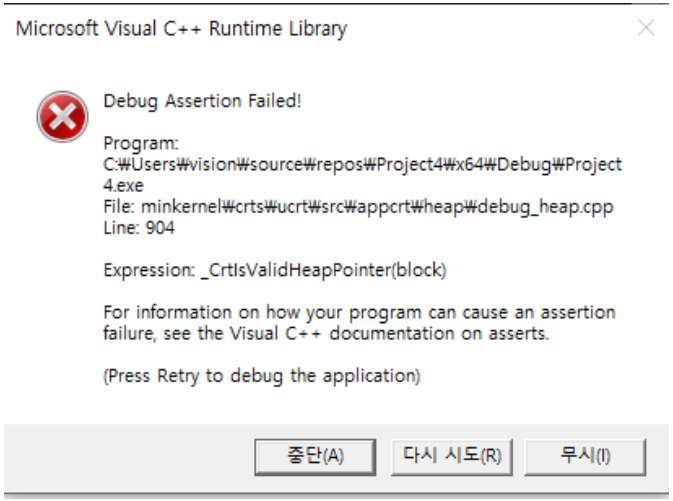
```
T* pointer = new T;
```

delete p 를 하게 되면 p 에 할당된 공간이 해제

포인터

```
int main() {
        int a = 5;
        delete& a;
        return 0;
}
```

지역 변수를 무리하게 delete 로 해제



new 로 배열 할당

SEOIL UNIVERSITY

```
int main() {
        int arr_size;
        cout << "array size : ";
        cin >> arr_size;
        int* list = new int[arr_size];

        for (int i = 0; i < arr_size; i++) {
                cin >> list[i];
        }
        for (int i = 0; i < arr_size; i++) {
                cout << i << "th element of list : " << list[i] << endl;
        }
        delete[] list;
        return 0;
}
```

```
        delete[] list;
        for (int i = 0; i < arr_size; i++) {
                cout << i << "th element of list : " << list[i] << endl;
        }
```

new 로 배열 할당

list 에 new 를 이용하여 크기가 arr_size 인 int 배열을 생성

```
T* pointer = new T[size];
```

```
delete[] list;
```

delete [] 를 통해서 해제
new – delete 가 짝을 이루고
new [] 와delete [] 가 짝

감사합니다.