

* pointer

SCP_이예준

내부세미나



목차

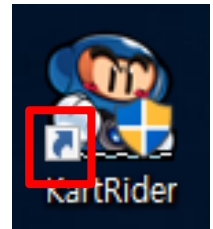
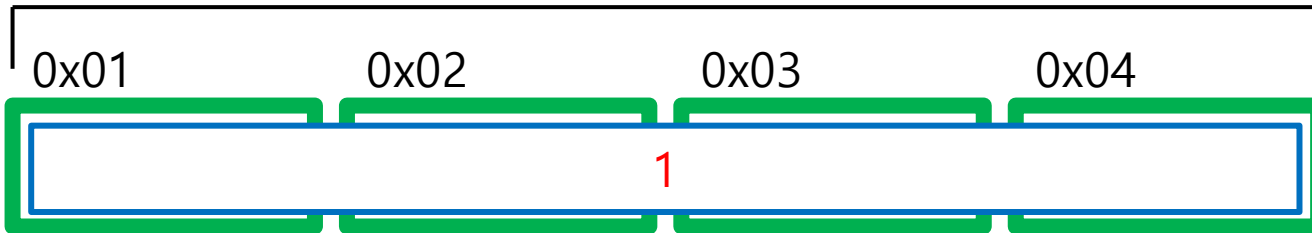
- 포인터란?
- 포인터 선언
- 포인터 연산자
- 메모리 구조
- 포인터 변수 크기
- 포인터 자료형
- 간접 접근
- 배열과 포인터
- 함수와 포인터
- 포인터의 단점

포인터란?

메모리 주소를 '**참조**' 하는 녀석

```
int a=1; // int -> 4byte
```

int a

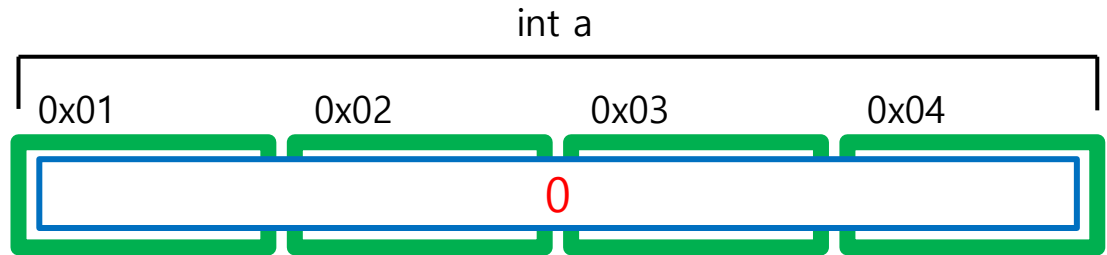


// 시작주소를 가리킨다.

a를 참조하는
포인터

포인터 선언

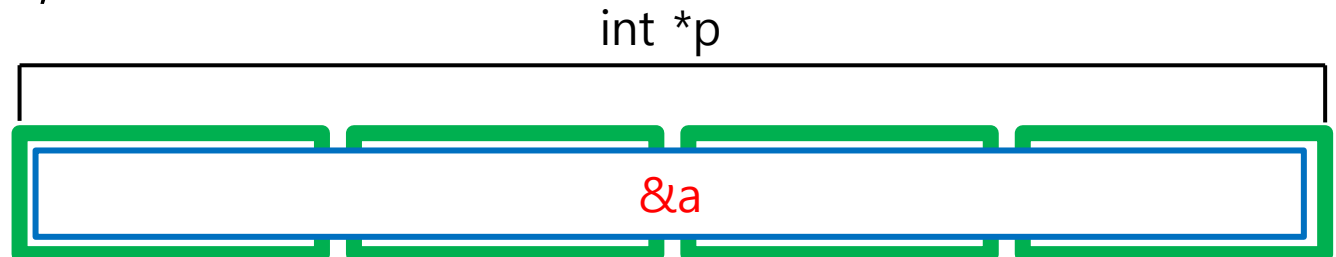
① `int a=0;` // 일반 변수



③ `p=&a`

// 시작주소를 가리킨다.

② `int *p=NULL;` // 포인터 변수



포인터 연산자 (&, *)

& : 변수 또는 상수의 주소
* : 포인터가 가리키는 주소에 저장된 값

```
#include<stdio.h>
```

```
int main() {  
    int a = 0;  
    int* p = &a;  
  
    *p = 10; // *(&a)=10; = a=10;  
  
    printf("a의 값 = %d\n", a);  
    printf("a의 값 = %d\n", *p);  
  
    printf("p의 값 = %x\n", p);  
    printf("a의 주소 = %x\n", &a);  
  
    printf("p의 주소 = %x\n", &p);  
  
    return 0;  
}
```



실행결과

```
a의 값 = 10  
a의 값 = 10  
  
p의 값 = 95fe9c  
a의 주소 = 95fe9c  
  
p의 주소 = 95fe90  
  
계속하려면 아무 키나 누르십시오...
```

메모리 구조

```
int a = 0;  
int* p = &a;
```

실행결과

```
a의 값 = 10  
a의 값 = 10  
p의 값 = 95fe9c  
a의 주소 = 95fe9c // C(16) -> 12(10)  
p의 주소 = 95fe90
```

계속하려면 아무 키나 누르십시오

Stack

Heap

Data

Text

높은 주소

- ① 0x95fe9c // 변수a의 주소
- ② 0x95fe90 // 변수p의 주소

③ c

② b

① a

⋮

⋮

낮은 주소

포인터 변수 크기

```
#include<stdio.h>
```

```
int main() {  
    int a1 = 12;  
  
    char* p1 = &a1;  
    int* p2 = &a1;  
    double* p3 = &a1;  
  
    printf("a1변수의 크기 = %d\n", sizeof(a1));  
    printf("p1변수의 크기 = %d\n", sizeof(p1));  
    printf("p2변수의 크기 = %d\n", sizeof(p2));  
    printf("p3변수의 크기 = %d\n", sizeof(p3));  
    return 0;  
}
```

실행결과

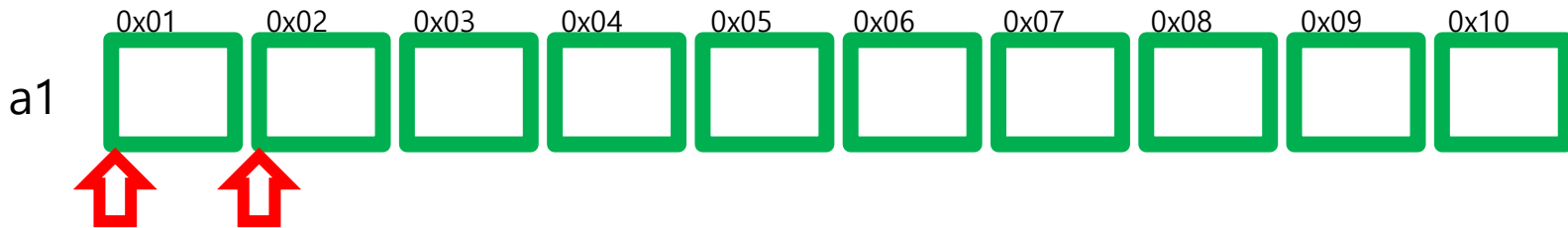
```
a1변수의 크기 = 4  
p1변수의 크기 = 4  
p2변수의 크기 = 4  
p3변수의 크기 = 4  
계속하려면 아무 키나 누르십
```

포인터 변수는 **자료형**에 상관 없이
모두 **4byte**의 크기를 가진다.

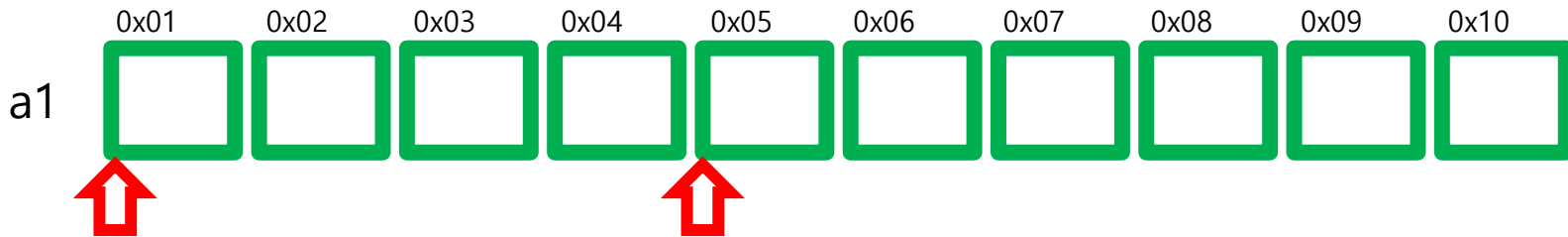
포인터 자료형

```
char* p1 = &a1;  
int* p2 = &a1;  
double* p3 = &a1;
```

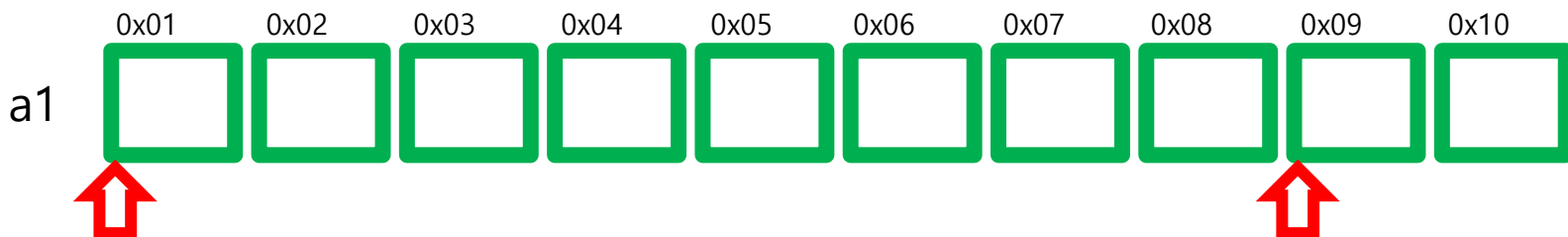
char* p1 + 1



int* p2 + 1



double* p3 + 1



간접 접근

실행결과

```
#include<stdio.h>

int main() {

    char a = 'A';
    int b = 10;
    char* p = &b;

    printf("%x\n", &a);
    printf("%x -> %x\n", p, (p+15));
    printf("%c\n", *(&a));
    printf("%d -> %c\n", *p, *(p+15));

    return 0;
}
```

8ff7af
8ff7a0 -> 8ff7af
A
10 -> A
계속하려면 아무 키나

주소 값 차이 15

char(1byte)*15

간접 접근 가능

간접 접근

```
#include<stdio.h>

int main() {

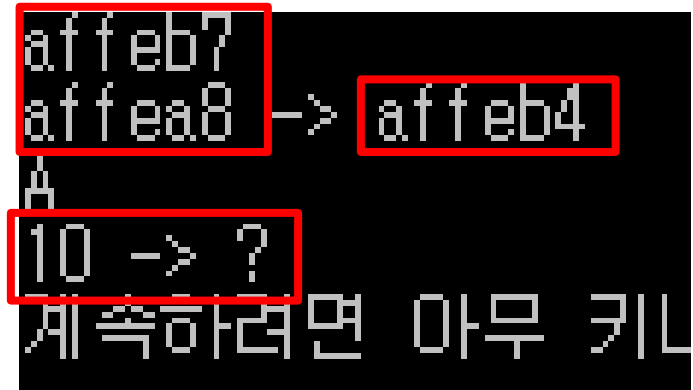
    char a = 'A';
    int b = 10;
    int* p = &b;

    printf("%x\n", &a);
    printf("%x -> %x\n", p, (p+3));

    printf("%c\n", *(&a));
    printf("%d -> %c\n", *p, *(p+3));

    return 0;
}
```

실행결과



affeb7
affea8 -> affeb4
A
10 -> ?
계속하려면 아무 키나

주소 값 차이 15

$\text{int}(4\text{byte}) * 3 = 12$

$\text{int}(4\text{byte}) * 4 = 16$

간접 접근 불가

※ 괄호()의 중요성

```
#include<stdio.h>
```

```
int main() {
```

```
    int a[5] = { 10,20,30,40 };
```

```
    int* p = a;
```

```
    printf("%d -> %d\n", *p, *p + 3);
```

```
    printf("%d -> %d\n", *p, *(p+3));
```

```
    return 0;
```

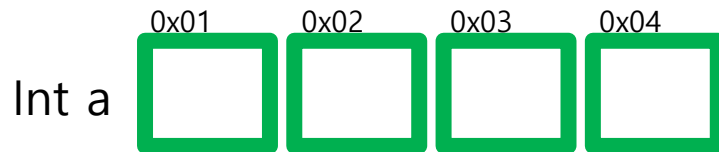
```
}
```

실행결과

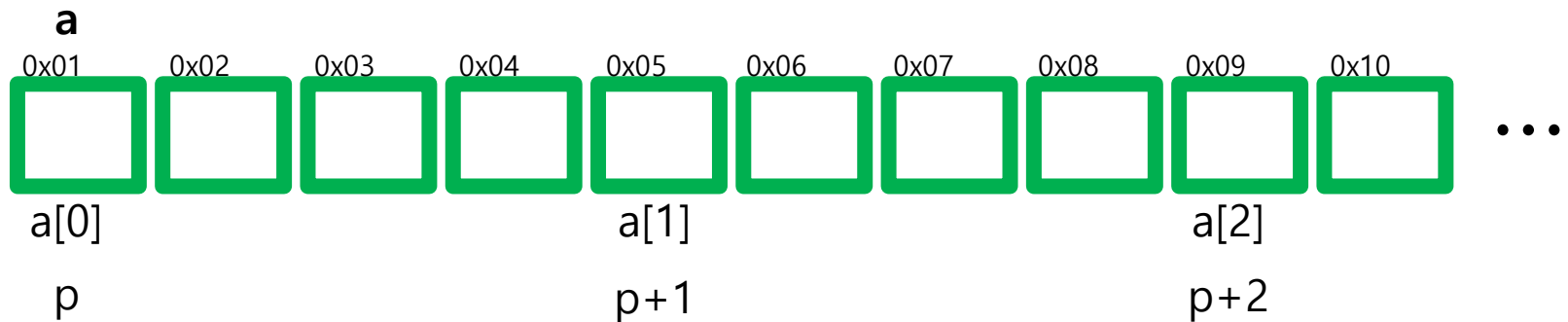
```
10 -> 13
10 -> 40
```

변수와 배열

변수



배열 ex) int형

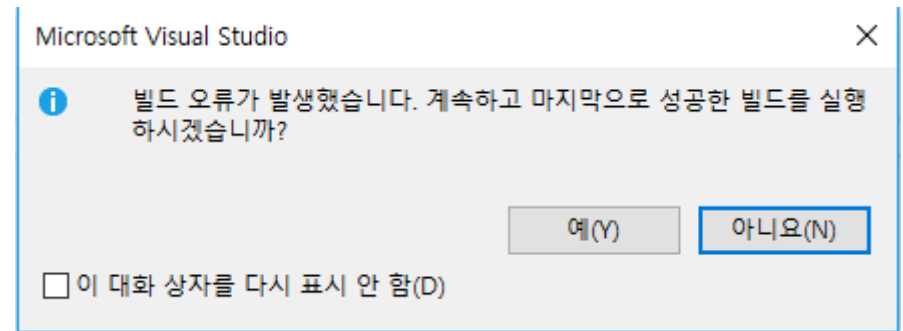


Why?

배열과 포인터

- 메모리의 효율성

```
#include<stdio.h>
int main() {
    int A, B, T, i;
    scanf("%d", &T);
    int result[T];
    for (i = 0; i <= T - 1; i++) {
        scanf("%d %d", &A, &B);
        result[i] = A + B;
    }
    for (i = 0; i <= T - 1; i++) {
        printf("%d\\n", result[i]);
    }
    return 0;
}
```



배열 -> 동적 할당 X

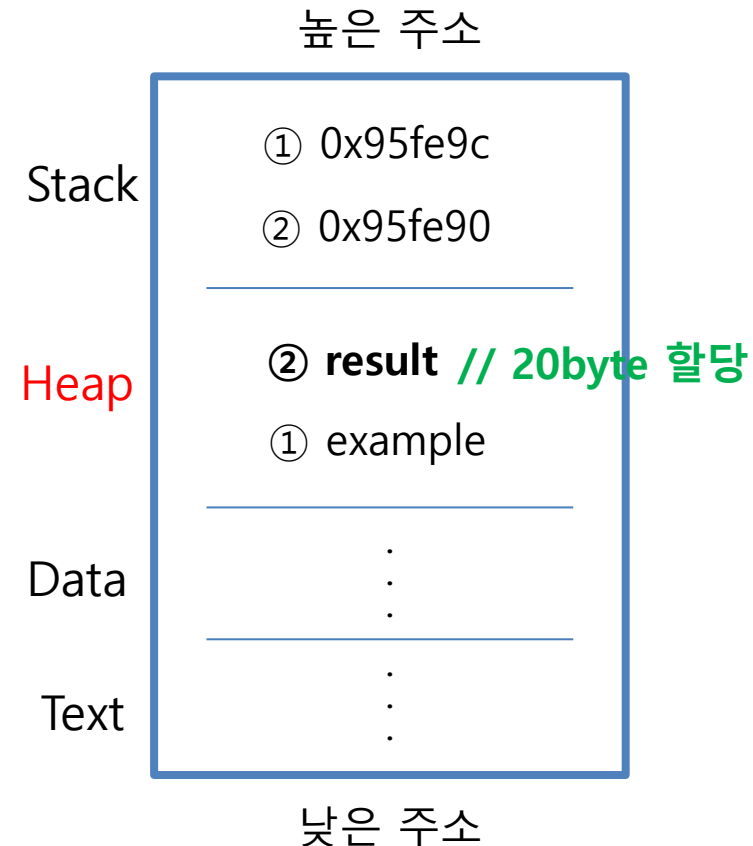
배열과 포인터

- 메모리의 효율성

```
#include<stdio.h>
#include<stdlib.h>

int main() {
    int A, B, i, T;
    scanf("%d", &T); // ex) 5 입력
    int* result = (int*)malloc(sizeof(int)*T);
    for (i = 0; i <= T - 1; i++) {
        scanf("%d %d", &A, &B);
        result[i] = A + B;
    }
    for (i = 0; i <= T - 1; i++) {
        printf("%d\\n", result[i]);
    }
    return 0;
}
```

포인터 -> 동적 할당 O



배열과 포인터

- 메모리의 효율성

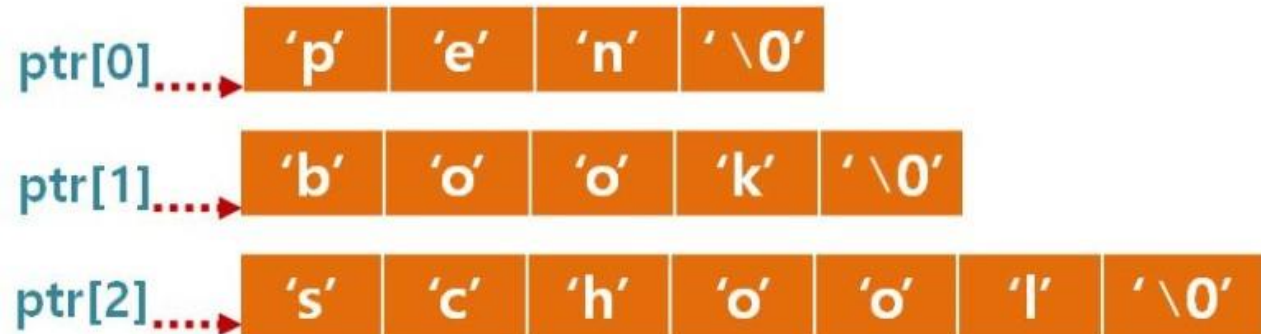
2차원 배열

```
char arr[3][7] = {"pen", "book", "school"};
```



```
char *ptr[3] = {"pen", "book", "school"};
```

포인터 배열



함수와 포인터

- Call by Value

```
#include<stdio.h>
void swap(int argA, int argB);
int main() {
    int A = 10, B = 20;
    swap(A, B);
    printf("A: %d, B: %d\n", A, B);
    return 0;
}
```

```
void swap(int argA, int argB) {
    int temp;

    temp = argA;
    argA = argB;
    argB = temp;

    return 0;
}
```

실행결과

```
A: 10, B: 20
계속하려면 아무
```

// int argA = A(10);
// int argB = B(20);

함수 내부의 지역변수



main()에 A와 B에게 영향X

함수와 포인터

- Call by Reference

```
#include<stdio.h>
void swap(int *argA, int *argB);
int main() {
    int A = 10, B = 20;
    swap(&A, &B);
    printf("A: %d, B: %d\n", A, B);
    return 0;
}

void swap(int *argA, int *argB) {
    int temp;

    temp = *argA;
    *argA = *argB;
    *argB = temp;

    return;
}
```

실행결과

```
A: 20, B: 10
계속하려면 아무 키를 누르세요
```

```
// int *argA = &A;
// int *argB = &B;
```

메모리 상의 주소 전달



원본의 값 수정 가능

포인터의 단점

- 포인터 변수는 **주소를 직접적으로 컨트롤**하기 때문에 예외 처리가 확실하지 않을 경우 **예상치 못한 문제가 많이 발생**.
(널 포인터 같은 경우에 바로 접근할 경우 예외 발생)
- 선언만 하고 **초기화를 하지 않을 경우**
쓰레기 주소를 가리키고 있기 때문에 사용에 주의
- 포인터 변수는 **주소를 직접 참조**하기 때문에
의도하지 않게 원본의 값이 수정 될 수 있다.
- 오류를 범하기 쉽고 기교적인 프로그램이 되기 쉽다.
- 프로그램의 이해와 버그 찾기가 어렵다.
- 메모리 절대 번지 접근 시 **시스템 오류를 초래**한다.

***감사합니다**

