Computer Architecture & Real-Time Operating System

2. C Pointers

Prof. Jong-Chan Kim

Dept. Automobile and IT Convergence

KMU 국민대학교 KOOKMIN UNIVERSITY

Array

- A collection of objects
  - of a same data type
  - stored in a contiguous memory area
  - represented by a given array name
  - indexed by an integer (0, 1, 2, …)

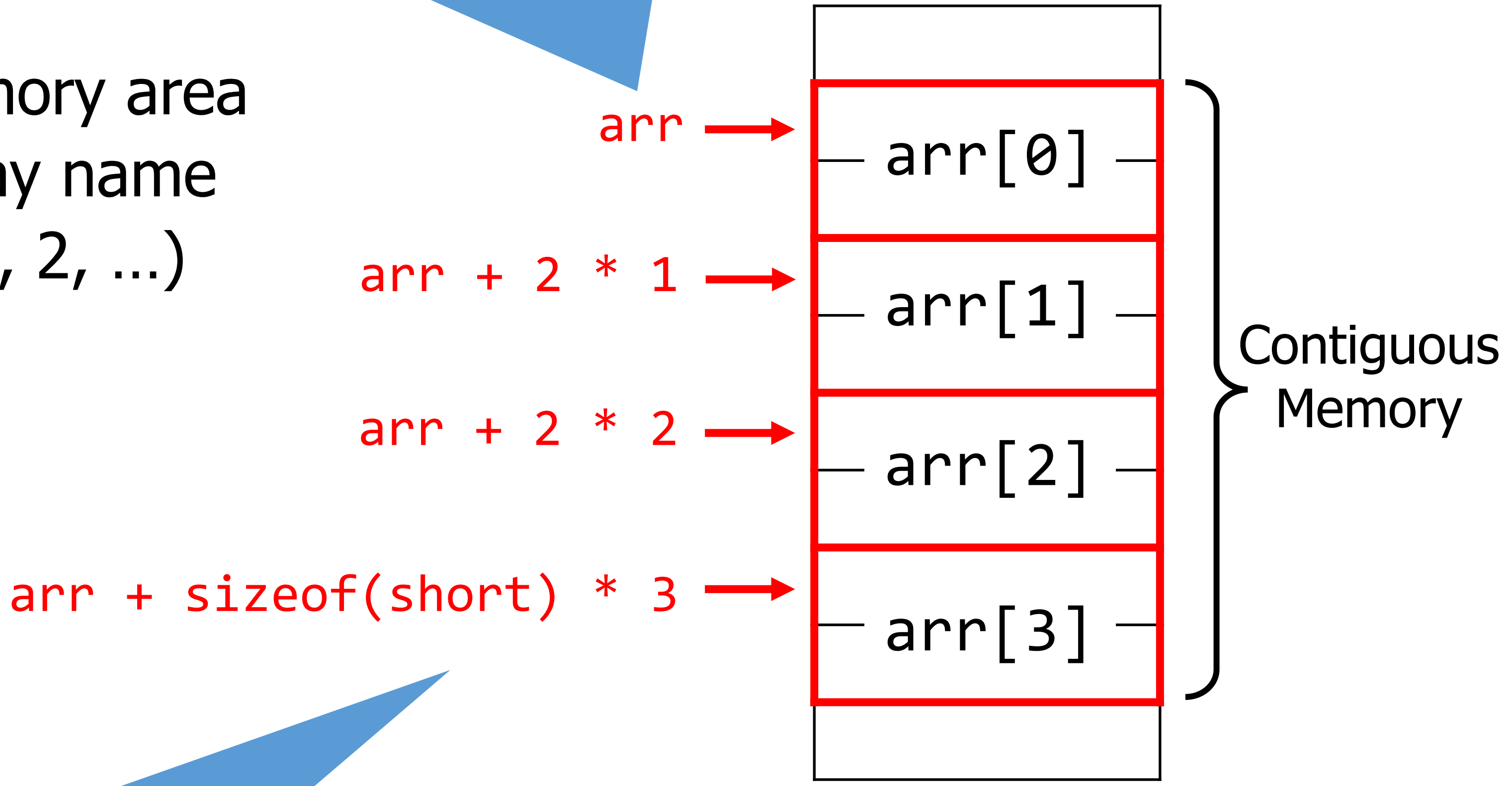
Type of each element

Number of elements

```
short arr[4];
```

Array name

Array name denotes the starting address



Memory layout
(Each cell means a single byte)

`arr[i]` is at `arr + sizeof(arr[0]) * i`

Array Examples

```
int A[10];
```

- An array of 10 integers
- `A[0], A[1], …, A[9]`

```
double B[20];
```

- An array of 20 doubles
- `B[0], B[1], …, B[19]`

Array Initialization

```
int A[5] = {1, 3, 5, 7, 9};
```

- Five elements all with initial values

```
int B[20] = {1, 3, 5, 7, 9};
```

- Partially initialized with the remaining implicitly initialized as zeroes

```
int C[] = {1, 3, 5, 7, 9};
```

- Array size is automatically determined by the number of initial values

Size of Array

Try size.c

- `sizeof(array)` gives the array's total size in bytes
- `sizeof(array[0])` gives a single element's size in bytes

```
int A[5];

printf("%ld\n", sizeof(A));
printf("%ld\n", sizeof(A[0]));
printf("%ld\n", sizeof(A) / sizeof(A[0]));
```

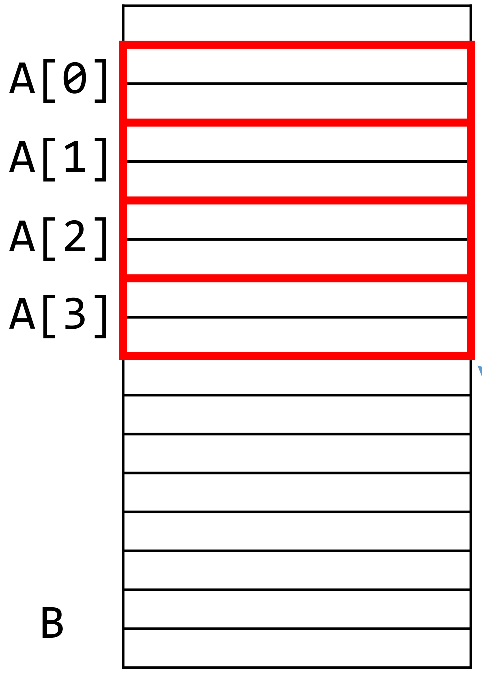
Array Indexing

Try index.c

- Index begins with 0
- C does not check the array index bound
- It is the programmer's responsibility not to go over
- If it happens, it is called "array overflow" that can cause disasters

```
short A[4];
short B = 0;
A[7] = 10;
```

What happens?



Note: memory layout can be different for different compilers and OSes

Two-dimensional Array

```
short A[2][3];
```

Number of rows Number of columns

2 x 3 array:

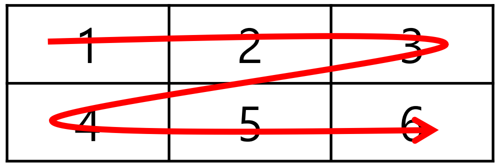
1	2	3
4	5	6

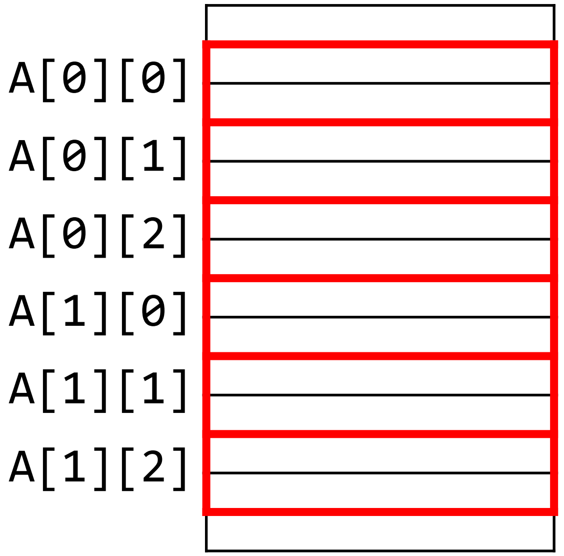
```
short i, j;
short A[2][3] = {{1, 2, 3}, {4, 5, 6}};

for (i = 0; i < 2; i++) {
    for (j = 0; j < 3; j++) {
        printf("%d ", A[i][j]);
    }
    printf("\n");
}
```

Row index Column index

Two-dimensional Arrays in Memory

2 x 3 array:





`A[i][j]` is located at `A + sizeof(A[0]) * i + sizeof(A[0][0]) * j`

Size of a row

Size of an element

Array as a Function Parameter

- How to pass an array to a function?

```
short arr[5] = {3, 9, 1, 5, 7};

show(arr);
```

```
void show(short arr[5])
{
    int i;
    for (i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    return;
}
```

Array as a Function Parameter

- How to pass an array to a function?

```
short arr[5] = {3, 9, 1, 5, 7};
show(arr);
short arr2[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
show(arr2);
```

```
void show(short arr[5])
{
    int i;
    for (i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    return;
}
```

This function cannot be generally used for various array sizes

Array as a Function Parameter

Try pass.c

- How to pass an array to a function?

```
short arr[5] = {3, 9, 1, 5, 7};

show(arr, 5);
```

```
void show(short arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return;
}
```

Solution:
The array size should be passed as a separate function parameter

Array as a Function Parameter

- What happens if the passed array is modified in the function? Will it be kept after returning from the function or not?

```
short arr[5] = {3, 9, 1, 5, 7};

show(arr, 5);
```

```
void show(short arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        arr[i] = 0;
    return;
}
```

Power of Pointers

- The unique power of the C programming language
- Higher-level languages like Python and Java do not have this power
- Pointers allow your program to access any memory location

Source: Avengers Endgame

A single illegal pointer access can instantly destroy the entire system like the Infinity Gauntlet

Memory Address

Address

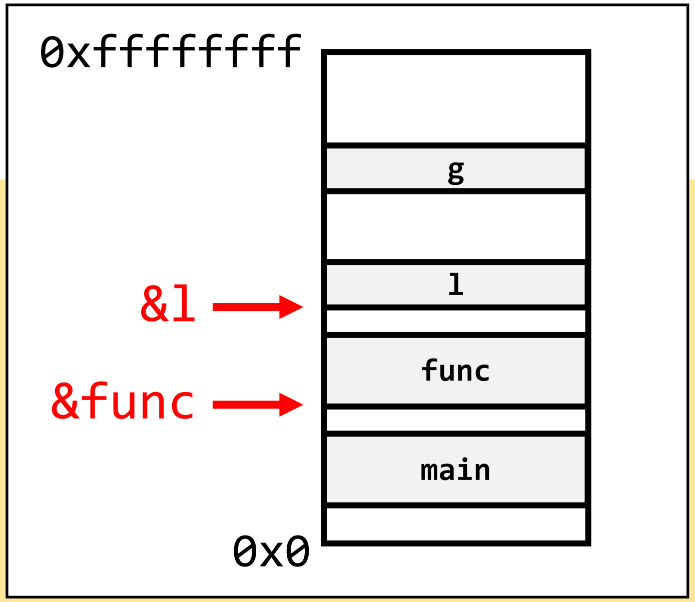
Try memory.c

- Variables and functions have their memory locations
- &: "address-of" operator

```
#include <stdio.h>
int g = 0;
void func(void)
{
    return;
}
int main(void)
{
    int l = 0;
    printf("%p %p %p %p\n", &g, &func, &main, &l);
    return 0;
}
```

Addresses of variables and functions

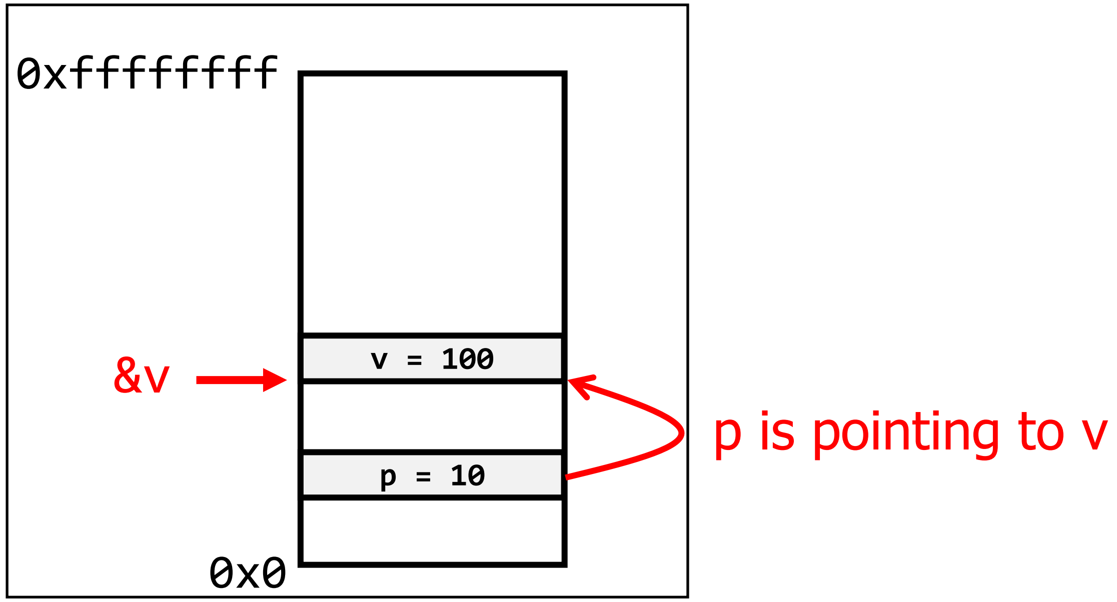
%p means a pointer (or an address)



Pointer Variables

- A variable containing an address of an object (i.e., variable or function)

```
int v = 100;

int *p;

p = &v;
```



Declaring a Pointer Variable

- A pointer variable p containing the address of a data of a type

```
type *p;
```

```
int *p;    /* a pointer to an int */

double *q; /* a pointer to a double */

char *r;   /* a pointer to a char */

void *s;   /* a pointer without an associated type */
```

Q: Why a data type is given when declaring a pointer variable?

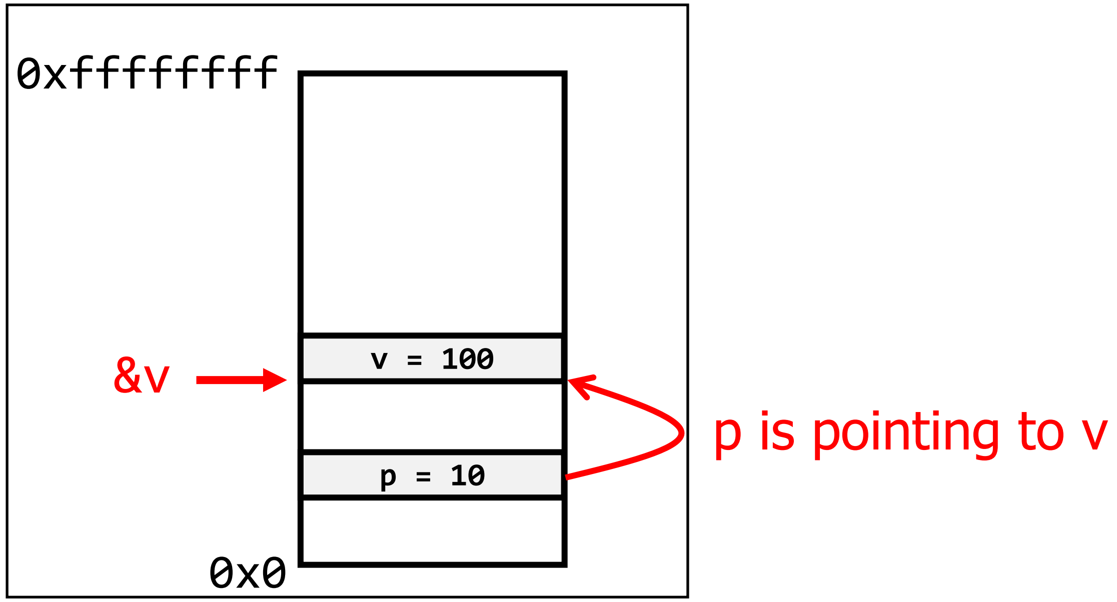
Dereferencing a Pointer

- Use '*' – the dereference operator – to a pointer to get the value of the pointed variable

```
int v = 100;
int *p;
p = &v;
printf("%d %d", v, *p);
```

the same as v

p should be initialized before being dereferenced. If not, what happens?



A: To determine the size of the pointed data and properly interpret it

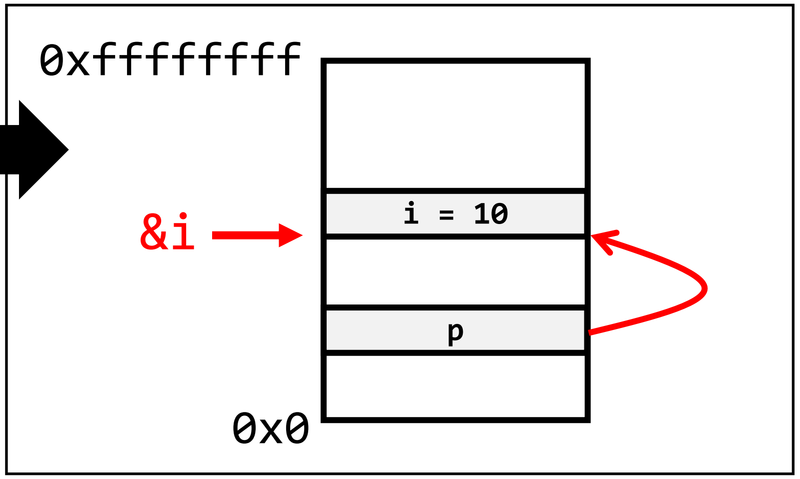
Not knowing the data type, we cannot interpret it

Why Type Matters?

- Using a mismatched pointer type can be catastrophic

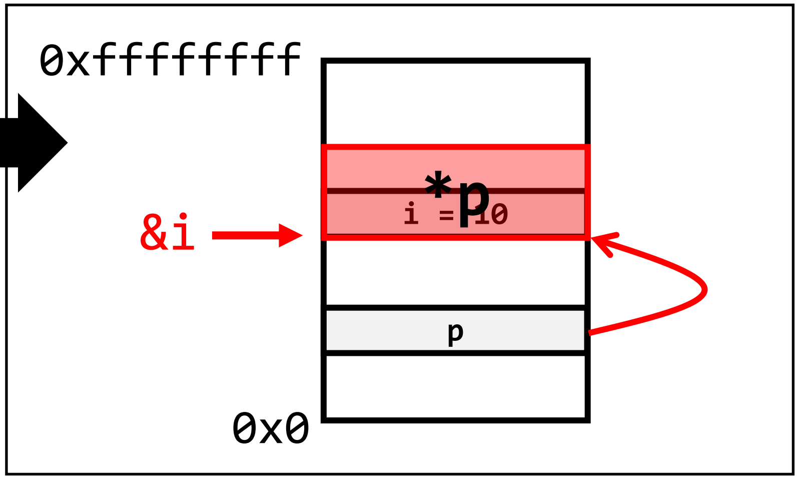
4 bytes

```
int i = 10;
int *p = &i;
*p += 10;
printf("%d\n", i);
```



8 bytes

```
int i = 10;
double *p = &i;
*p += 10;
printf("%d\n", i);
```

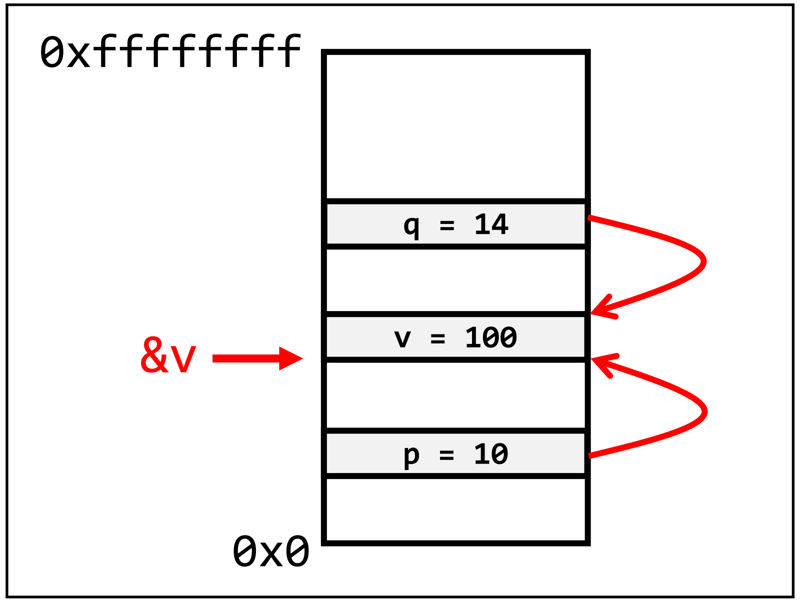


Pointer Arithmetic

- Adding or subtracting an integer value to and from a pointer variable

```
int v = 100;
int *p = &v;
int *q = p + 1;
printf("%p %p\n", p, q);
```

Where is q pointing to now?



- When doing arithmetic with pointers, 1 does not mean 1
- Instead, 1 means `sizeof(type)`

Array and Pointers

- Arrays and pointers are fundamentally the same thing
  - `int A[5]`
  - `int *p`

A and **p** are of the same type **int ***

Pointer to an integer

```
int A[5] = {1, 2, 3, 4, 5};
int *p = A;

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

The only difference is
- A cannot be changed
- p can be assigned with another value

Array Indexing and Pointer Arithmetic Try arith.c

- Assuming `A` is an array name
  - `*A           := A[0]`
  - `*(A + i)     := A[i]`
  - `A + i        := &A[i] = A + sizeof(A[0]) * i`

```
int A[5] = {1, 2, 3, 4, 5};
int i;

printf("*A, A[0]: %d %d\n", *A, A[0]);

for (i = 0; i < 5; i++) {
    printf("====================\n");
    printf("*(A + %d), A[%d]: %d %d\n", i, i, *(A + i), A[i]);
    printf("  A + %d, &A[%d]: %p %p\n", i, i, A + i, &A[i]);
}
```

Passing an Array via a Pointer

```c
void show(short arr[], int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return;
}
```

Do the same thing

```c
void show(short *arr, int n)
{
    int i;
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    return;
}
```

Homework

- Read
  - http://csapp.cs.cmu.edu/2e/ch1-preview.pdf

- Finish setting up your programming environment until the next lecture

Questions