UNIVERSITY OF MASSACHUSETTS
DARTMOUTH

ECE160: Foundations of Computer Engineering I

Lecture #28 –Pointers and Arrays

Instructor: Dr. Liudong Xing
SENG-213C, lxing@umassd.edu
ECE Dept.

Administrative Issues

- Lab #12 (Review Exam#3) due **5pm, April 26**
- Final exam on Monday, May 1 (8am-11am)
- Today's topics:
  - Discuss Exam#3 solutions
  - Lecture #28 (pointers and arrays)

Review of Lecture #26

- Using pointers
  - to increment a number
  - to test for equality using pointers
  - to add two numbers
- Use multiple pointers for one variable
- Use pointers that point to other pointers
- Pointers and functions
  - Pointers can be arguments to a function (pass by reference)
  - Pointers can be returned from a function

Arrays and Pointers

- Arrays and pointers have a very close relationship
  - The array name is a pointer constant to the first element of the array
  - We can use array name anywhere we can use a pointer, specifically, with the indirection operator *

```
int a[4] = {1,10,30,4};
int *p = a;
```

- Given pointer p, p±n is a pointer to the value *n* elements away
  - If p is a pointer pointing to the second element of an array
  - p-1 is a pointer to the previous (first) element
  - p+1 is a pointer to the next (third) element

An Example

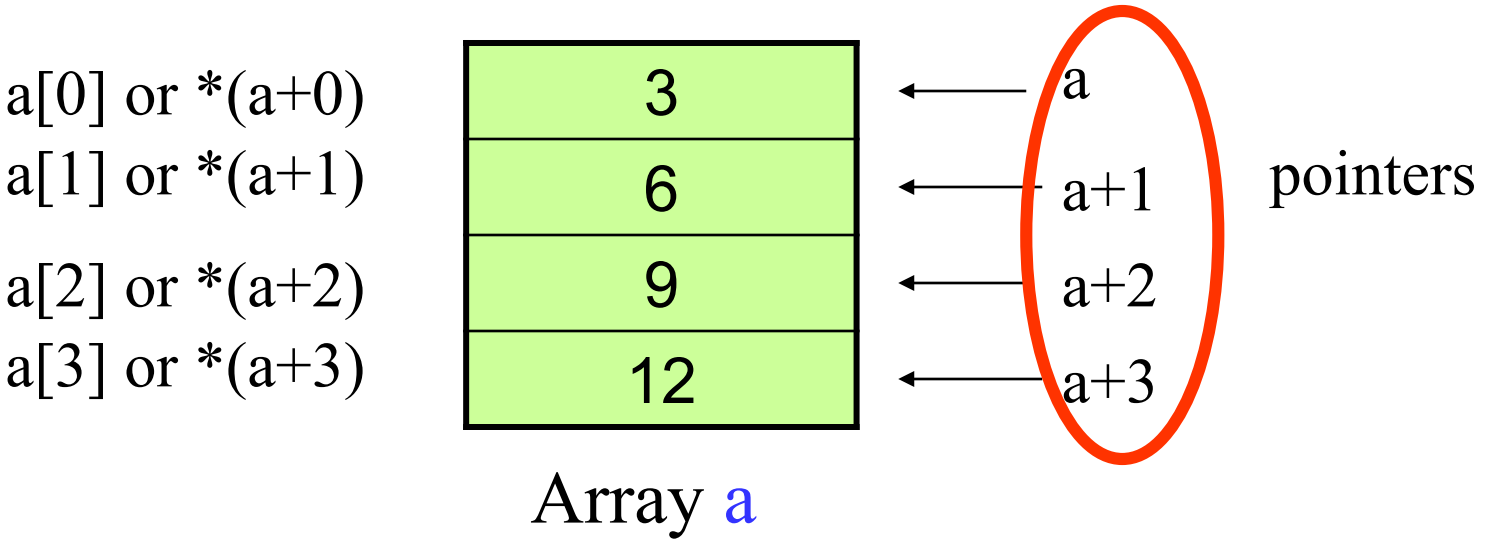
```
#include "stdio.h"

void main(void)
{
        int a[4] = {1,10,30,4};
        int *p = a;

        printf("The address is %p\n", &a[0]);
        printf("The address is %p\n", a);
        printf("The element is %d\n", a[2]);
        printf("The element is %d\n", *(p+2));
        printf("The element is %d\n", *(a+2));

}
```

What is the output?

Note!

- The following two expressions are exactly the same when a is the name of an array and n is an integer:

*(a+n) is identical to a[n]



Exercise

- Write a program that adds 300 to all elements of an array that has 6 elements and prints the new array. It uses a pointer to access the elements of the array.

```
#include "stdio.h"
void main(void)
{
   int a[6] = {1, 10, 30, 4, 6, 67};
   int *p = a;
   int i;

   //add 300 to each element and print it out
                          ???

}
```

Passing an Array to a Function

- In the called function prototype and definition header
  - Way 1: use the traditional array notation to indicate that the parameter is an array:

    ```
    int my_func(int a[]);
    ```

  - Way 2: use pointers:

    ```
    int my_func(int *a);
    ```

- In the calling function, use the array name as the parameter in the function call

Modification Exercise

- Write a program that amplifies each element of an array with 4 elements by 100 and then prints the new array. Call a function to do the multiplication part!

```
#include "stdio.h"
void multiply(int a[]);

void main(void)
{
int arr[4] = {10,20,30,40};
int i;

//call the multiply function here
multiply(arr);

//output the amplified array elements
for (i = 0; i < 4; i++)
{
   printf("%d\n", arr[i]);
}
}
```

Change to using pointers!

```
//function definition
void multiply(int a[])
{
for (int i = 0; i < 4; i++)
{
   a[i] = a[i] * 100;
}
}
```

Pointer Compatibility

- Pointers have a type associated with them
- The types are not just pointer types, but rather are pointers to a specific type, such as int, char

- Pointer types must match, otherwise, using a cast operator (Lecture#9) so that you can make an explicit assignment between incompatible pointer types!

Example (1)

```
char c='a';
int a=0;

char *pc;

pc = &a;
/*invalid*/

pc = (char*) &a;
/*valid: use a cast operator (new type) to cast &a to a char pointer*/
```

- It's invalid to assign a pointer of one type to a pointer of another type, even though the values in both cases are memory addresses and would therefore seem to be fully compatible!
- Use a cast operator so that you can make an explicit assignment between incompatible pointer types!

Note!

- In C, a lower order type is automatically converted/promoted to a higher order type (Lecture#9), but this does not apply to pointers.
- For example: we can say:

```
int a;
char c = 'A';
a = c;
```

The char would be converted to an integer value first and then assignment

but we can't say:

```
int* a;
char c = 'A';
a = &c;
```

instead we have to say:

```
int* a;
char c = 'A';
a = (int*) &c;
```

Promotion Hierarchy (L#9, revisit)

Highest →	long double
	double
	float
	unsigned long int
	long int
	unsigned int
	int
	short
Lowest →	char

Example (2)

```
#include "stdio.h"
void main(void)
{
int x = 66;
int* px;
char c = 'A';
char* pc;

printf("The size of x is: %d\n", sizeof(x));
printf("The size of px is: %d\n", sizeof(px));
printf("The size of c is: %d\n", sizeof(c));
printf("The size of pc is: %d\n", sizeof(pc));

px = &x;
pc = (char*)&x;
px = (int*)&c;
printf("x is %d\n", *pc);
printf("c is %c\n", *px);
}
```

- sizeof(): tells the size in bytes of the operand
- Assume
  - the size of an integer is 4
  - the size of a char is 1 byte
  - The size of an address is 4 bytes
- What is the output of the program?
- What happens if removing (char*) and (int*)?

Summary of Lecture #28

- The array name is a pointer constant to the first element of the array
- We can use array name anywhere we can use a pointer, specifically, we can use the array name with the indirection operator *
- We can pass the whole array to a function
- Pointer types must match, otherwise, using a cast operator so that you can make an explicit assignment between incompatible pointer types!

Things To Do

- Complete Lab #12 (Review Exam#3) due **5pm, April 26**

- Review lecture notes, lab and homework problems to prepare for the final exam