EEE101: C Programming & Software Engineering I

Lecture 6: Arrays and Pointers 1

Dr. Rui Lin/Dr. Mark Leach
Office: EE512/EE510
Email: rui.lin/mark.leach@xjtlu.edu.cn
Dept. of EEE XJTLU

Outline of Today's Lecture (6)

- One-dimensional arrays
- Multi-dimensional arrays
- Arrays and loops
- Introduction to pointers
- Pointer Arithmetic

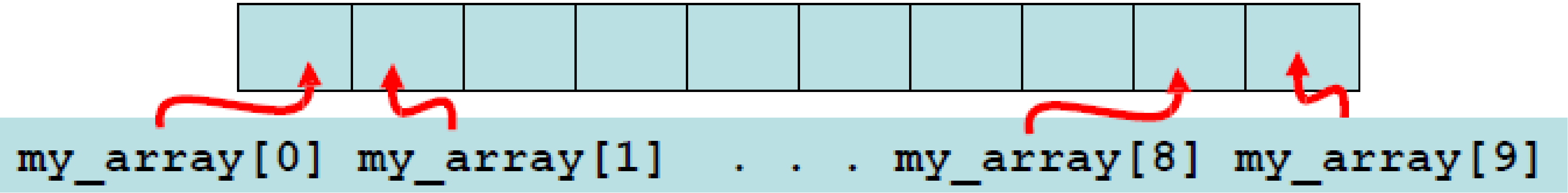
Arrays – An Introduction (1/2)

- In C, an array is essentially a group of elements all belonging to the **same type**, having the **same name.**
- Array elements are arranged **sequentially in the memory** space, but you can access them in any order.
- Each element of an array is accessed using an **index** (an integer). The **index range** is from **zero** to **N-1**, where N is the array length.
- Arrays are commonly linked to pointers

Arrays – An Introduction (2/2)

Consider a 10 element array of integers called my_array:

```
int my_array[10];
```



my_array[0] my_array[1] . . . my_array[8] my_array[9]

my_array[0] – is the name of the first array element

my_array[9] – is the name of the 10th array element

my_array[10] – is outside of the array bounds

depending on the program this may cause a runtime error.

Declaring and Initialising Arrays (1/2)

- Arrays occupy space in the computer's memory
- Specifying the **array type** and the **number of elements** determines the amount of memory

```
int my_array[10];
char x[100], y[20];
```

- Arrays can be initialised with a loop or at declaration:

```
for(i=0; i<10; i++){
        my_array[i]=0;
        }
```

```
int array[10]={0};
```

Declaring and Initialising Arrays (2/2)

- Arrays can be declared and initialised at the same time with the help of initialisers.

```
int my_array[10]={1,2,3,4,5,6,7,8,9,10};
float a[5]={3.0,2.0,3.0};
```

Any remaining elements are initialised to 0

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

array size is omitted, determined by initialised element number

- **The following causes a syntax error, Why?**

```
double x[5]={3.0,5.0,7.0,5.0,9.0,8.0};
```

- Initialisation of elements must begin at element 0.

Iterating through Array Elements

- A natural way to iterate through all of the elements of an array is to use a **for** loop:

```
int my_array[10]={1,2,3,4,5,6,7,8,9,10};
printf(“List all of the array elements”);
for(m=0;m<10;m++){
        printf(“%d \t”,my_array[m]);
        }
```

It is wrong to try to access array elements outside of the declared array length, however depending on your compiler there may be no error… **be careful!**

Arrays and Strings

- Character arrays can be initialised by individual characters:

```
char text[]={‘f’,’o’,’c’,’u’,’s’,’!’};
```

- They can also be initialised with a string:

```
char text[]={“focus!”};
```

Remember a string is a special character array ending in the NULL character ‘\0’.

Equivalent to:

```
char text[]={‘f’,’o’,’c’,’u’,’s’,’!’,’\0’};
```

Quick Quiz 1 – About Arrays

Which of the following statements is NOT true?

a) An array is a list of data elements stored consecutively in memory.

b) All the elements of an array are initially zero.

c) Array elements are accessed using an integer subscript.

d) Array elements can be modified using assignment statements.

e) Array elements must all be of the same data type.

Quick Quiz 1 – About Arrays

Which of the following statements is NOT true?

a) An array is a list of data elements stored consecutively in memory. **T**

b) All the elements of an array are initially zero. **F**

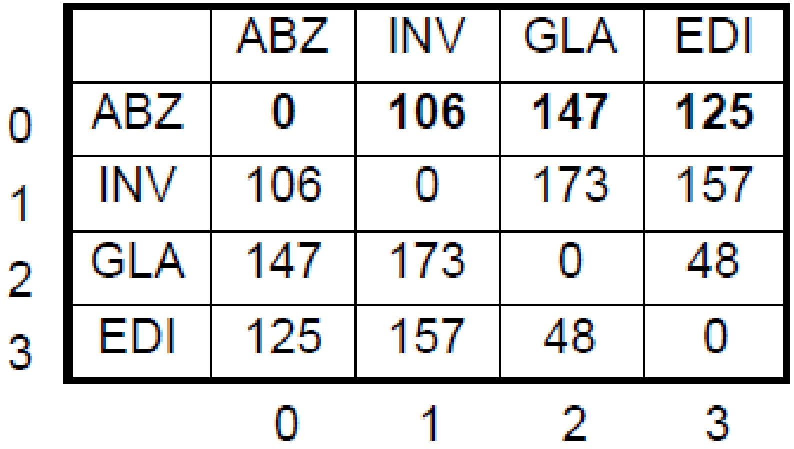
c) Array elements are accessed using an integer subscript. **T**

d) Array elements can be modified using assignment statements. **T**

e) Array elements must all be of the same data type. **T**

Two Dimensional Arrays

- In C, two-dimensional arrays have two subscripts, normally considered as: **[rowIndex][columnIndex]**



		ABZ	INV	GLA	EDI
0	ABZ	**0**	**106**	**147**	**125**
1	INV	106	0	173	157
2	GLA	147	173	0	48
3	EDI	125	157	48	0
		0	1	2	3

```
int miles[4][4] =
{ { 0, 106, 147, 125 },
{ 106, 0, 173, 157 },
{ 147, 173, 0, 48},
{ 125, 157, 48, 0} }; Or
int miles[][4]={0,106,147,125,106,0,173,
157,147,173,0,48,125,157,48,0};
```

Example distances between pairs of cities

- The first row is highlighted to show the element order [0][[0…3]
- The rowIndex can be left empty as long as initialisation is performed. The compiler can determine the number of rows.

Nested Loops and Arrays

- **Remember**, we looked at nested **for** loops:

```
char array[4][4];
for(c=0;c<4;c++){
        for(r=0;r<3;r++){
                array[r][c]=‘ ‘;
                }
        }
array[1][2] = ‘?’;
```

r	c
1	2

	0	1	2	3
0				
1			?	
2		?		

More Dimensions

You can define arrays with as many dimensions as you like

A two dimensional array looks like a table

A three dimensional array looks like a stack of tables

```
float table[3][5][2];
```

3 nested loops would be required to process through all of the elements in this array.

Quick Quiz 2 - Counting

What will be printed out?

a) 31

b) 59

c) 90

d) 120

e) 151

```
int days[]={31,28,31,30,31,30,
                  31,31,30,31,30,31};
int i=0, sum=0;
while(i<4)
         sum = sum + days[i++];
printf(“sum=%d\n”, sum);
```

Quick Quiz 2 - Counting

What will be printed out?

a) 31

b) 59

c) 90

d) 120

e) 151

```
int days[]={31,28,31,30,31,30,
                      31,31,30,31,30,31};
int i=0, sum=0;
while(i<4)
          sum = sum + days[i++];
printf(“sum=%d\n”, sum);
```

Pointers – An Introduction

Take a deep breath and prepare yourself…

Every **variable** is stored in the computers **memory** at a specific **address**. Just like you living in your home at your street address.

A **pointer** is a variable used to **store the address** of another variable.

Pointers – An Introduction

- Pointers are declared to **point at** a specified **type of variable** i.e. a **float** pointer must point at a **float** variable, an **int** pointer must point at an **int** variable

- Internally a pointer is stored as an **unsigned int**

- However you **cannot** use a pointer like an **int**

  e.g. you cannot multiply pointers

Pointer Declaration

Specify what type of variable the pointer points at

```
int *pAge;          /* pAge points at an int */
float *pHeight;     /* pHeight points at a float */
```

pAge and pHeight are pointers, they store addresses

int and **float** need different numbers of bytes (4 and 6)

Do you think pAge and pHeight need a different number of bytes?

Pointer Declaration

Specify what type of variable the pointer points at

```
int *pAge;          /* pAge points at an int */
float *pHeight;     /* pHeight points at a float */
```

pAge and pHeight at pointers, they store addresses

int and **float** need different numbers of bytes (4 and 6)

Do you think pAge and pHeight need a different number of bytes? **NO – they store the same thing!**

The Dereference Operator ‘*’

The * operator is used to find the value stored at an address

pAge is the pointer variable used to store an address

*pAge is the value stored at that address

If pAge points to an integer - ***pAge is the integer pointed at**

The Address of Operator ‘&’

Placing the ‘&’ operator in front of a variable finds the address of that variable.

If Age is a variable, &Age is the address of that variable.

scanf() makes use of the & operator to determine where to store the value entered in memory

```
int Age;
scanf(“%d”, &Age);
```

Using pointers and their op’s

Things you can do with a pointer:

Assignment (Value storage)

De-referencing (Value finding)

Taking the address of a pointer

Incrementing a pointer

Decrementing a pointer

Differencing two pointers

Pointer Assignments

Assigning an address to a pointer:

```
int x, y[3]={10,20,30}, *p1, *p2;

p1=&x;      /* assign address of x to p1  */
p2=y;       /* assign address of y[0] to p2 */
p2=&y[2];   /* assign address of y[2] to p2 */
```

- Note the array name ‘y’ is equivalent to &y[0]
- Also, notice the pointers are type **int** and point at **int** values

Pointer Address

A pointer is a variable…therefore it has it’s own address

```
printf(“%p”,&pArray);
                    /* print the address of a pointer */
```

- pArray is a pointer variable stored in the computer memory.
- The ‘&’ operator tells us where it is stored
- Format specifier %p %u or %lu can be used to print the value

De-referencing (Variables)

- Finding the value stored at another address ‘*’
- If the address of another variable is stored in a pointer, the pointer can access that variables value

```
int x=4, *p;
p=&x;                 /* assign the address of x to p */
printf(“%d”, *p);
/* print value stored at the address pointed at by
p, which is 4) */
*p=10;      /*place value 10 at memory address*/
printf(“%d”,x);
```

De-referencing (Arrays)

- What if the pointer is used to point to an array?

```
int x[5]={4,2,3,8,9}, *p;
p=x;
printf(“%d”, *p);
p=&x[2];
printf(“%d”, *p);
```

What values are printed here?

4 and 3 (x[0] and x[2])

Moving Pointers

- **Incrementing** (++) and **Decrementing** (--) is possible with pointers.
- The value contained in the pointer is an address. Incrementing and Decrementing **moves** the **number of bytes for** the **variable type**.

```
int x[5]={4,2,3,8,9}, *p;
p=x;
printf("%d %p\n", *p, p);
p++;
printf("%d %p", *p, p);
```

Q. If the address &x[0] = 0022FF1C and there are 4 bytes per integer value. What is printed on the screen?

Moving Pointers

Q. If the address &x[0] = 0022FF1C and there are 4 bytes per integer value. What is printed on the screen?

```
int x[5]={4,2,3,8,9}, *p;
p=x;                        \* assigns &x[0] to p *\
printf(“%d %p\n”, *p, p);
p++;
printf(“%d %p”, *p, p);
```

```
4      0022FF1C
2      0022FF20
```

Moving Pointers

Q. If the address &x[0] = 0022FF1C and there are 4 bytes per integer value. What is printed on the screen?

```
int x[5]={4,2,3,8,9}, *p;
p=x;                    \* assigns &x[0] to p *\
printf(“%d %p\n”, *p, p);
p++;
printf(“%d %p %p ”, *p, p, &p);
```

If I include the value &p what else is printed?

The address of variable p (generally unknown)

Moving Pointers

Be careful with precedence when working with pointers:

***p++ and (*p)++** are **not the same**

* and ++ are both unary operators the precedence is from right to left

***p++** will get the value from the current address and then increment the pointer.

(*p)++ will get the value from the current address and add 1 to that value

Pointer Subtraction

Differencing (or subtracting) two pointers can be used to determine how many elements apart two elements are in an array.

```
int x[5]={4,2,3,8,9}, *p1, *p2;
p1=&x[0];
p2=&x[2];
printf(“%d %d”, *p1-*p2, p2-p1);
```

Only useful if pointers are pointing to elements in the same array

What is printed in this case?

The first value is 4-3=1

The second value is 2 because the addresses are 8 bytes apart which is equivalent to 2 integers sizes

!!!!!WARNING!!!!

NEVER dereference an uninitialized pointer:

```
int *ptr     /* this is an uninitialized pointer*/
```

ptr contains a random value

```
*ptr = 5;    /* this is a very bad error */
```

This is trying to store a value in an unknown location

It could overwrite important data!

Pointers and Arrays (1/3)

Consider the following code extract:

```
float b[5] = {1.0, 2.0, 3.0, 4.0, 5.0};
float *bPtr;
bPtr=&b[0];
```

- Array element **b[3]** can be referenced by the pointer as:
  - ***(bPtr+3)**
  - ***(b+3)**
- The address of **&b[3]** is equivalent to **bPtr+3**

Pointers and Arrays (2/3)

- **char** s[] and **char** *s are equivalent (for any type)
- The following are **valid** pointer operations:
  - Assignment of pointers to same type variables
  - Adding or subtracting an integer and a pointer
  - Subtracting or comparing two pointers to elements of the same array
  - Assigning or comparing to zero
- ALL other operations are illegal!
- Behaviour is undefined for operations on pointers to elements of different arrays

Pointers and Arrays (3/3)

- Pointers are strongly related to arrays in C
- Pointers provide a symbolic way to use addresses
- Pointers are the most efficient way to deal with arrays

```
float a[5];
printf(“a=%p\n  &a[0]=%p\n”, a, &a[0]);
```

These are the same, the name of the array **a** and the address of the first element **&a[0]**.

Why is it useful to use a pointer for an array?

Operation use (a++ illegal), processing efficiency

Pointers: Do’s and Don’ts

Consider:

```
int one[3];
int *ptr1, *ptr2;
```

Valid	Illegal
ptr1++;	one++;
ptr2 = ptr2 + 2;	ptr2 = ptr2 + ptr1;
ptr2 = one + 1;	ptr2 = one + ptr1;

Note: it is not legal to add, divide, multiply etc. two pointers or add a float or double, or assign a pointer of one type to a variable of another without a cast.

Quick Quiz 3

Which value does ptr point at after the code finishes?

```
int a = 7;
int *ptr;
ptr = &a;
a += 5;
printf(“Value pointed to is %d”, *ptr);
```

a) 7
b) 5
c) 12
d) 0
e) undefined

Quick Quiz 3

Which value does ptr point at after the code finishes?

```
int a = 7;
int *ptr;
ptr = &a;
a += 5;
printf(“Value pointed to is %d”, *ptr);
```

a) 7

b) 5

c) 12

d) 0

e) undefined

Quick Quiz 4

Which value does ptr point at after the code finishes?

```
int a = 7;
int *ptr;
ptr = &a;
*ptr += a++;
printf(“Value pointed to is %d”, *ptr);
```

a) 7
b) 8
c) 14
d) 15
e) undefined

Quick Quiz 4

Which value does ptr point at after the code finishes?

```
int a = 7;
int *ptr;
ptr = &a;
*ptr += a++;
printf(“Value pointed to is %d”, *ptr);
```

a) 7

b) 8

c) 14

d) <u>15</u>

e) undefined

Questions?

Keep attending the labs ☺
We’re here to help