CMSC 21
Fundamentals of Programming

2nd Semester 2011-2012

Arrays, Strings, Structures

STRUCTURED DATA TYPES

Structured Data Types

- Collection of simple data type values arranged in some manner to facilitate easier access
- Examples are arrays, strings and structures

ONE-DIMENSIONAL ARRAYS

Arrays

- An array is simply a collection of data of the same type
- It is referenced by a common name or identifier

Declaring Arrays

- Arrays are declared in the program in this way:

```
<data_type> <var_name>[size];

int numbers[10];
float decimal[50];
```

Declaring Arrays

- `<data_type>` is any valid variable type in C. It can be a char, float, int, pointer, structure, etc.
- `<var_name>` is any valid identifier in C
- `size` is the maximum number of elements/values that an array can hold

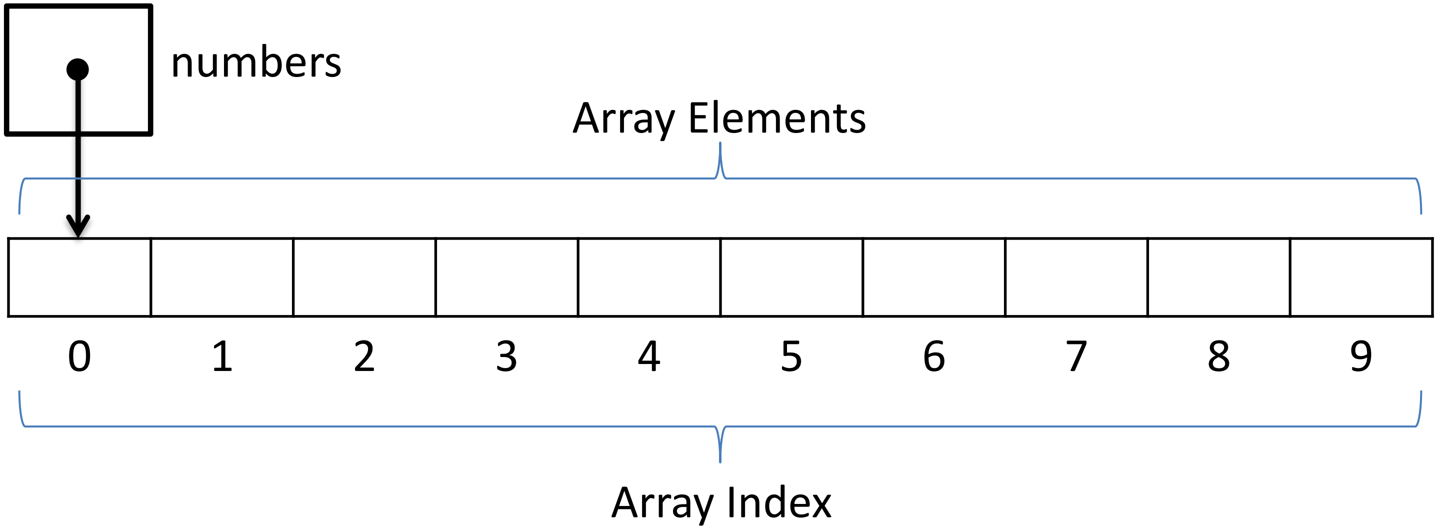
Arrays in the Memory

- When an array is declared, **consecutive memory locations** are reserved.
- The variable name is a **pointer (constant)** to the **first element** of the array
- Total space allocated for an array: consecutive memory locations equivalent to the size + a space for the pointer to the first element

Arrays in the Memory

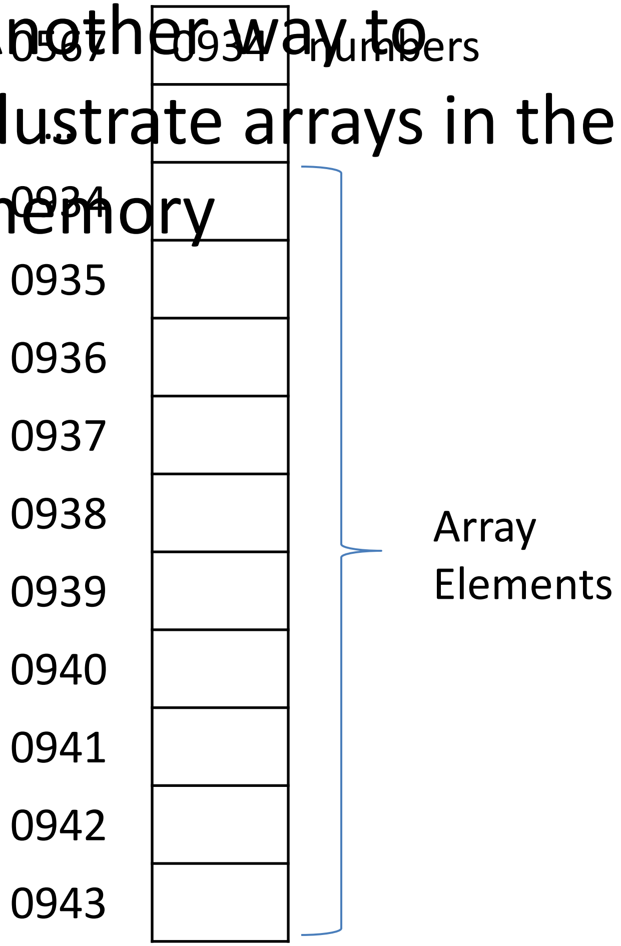
- An array can be illustrated as:

```
int numbers[10];
```



Arrays in the Memory

- Another way to illustrate arrays in the memory



0567 0934 numbers

...

0934

0935

0936

0937

0938

0939

0940

0941

0942

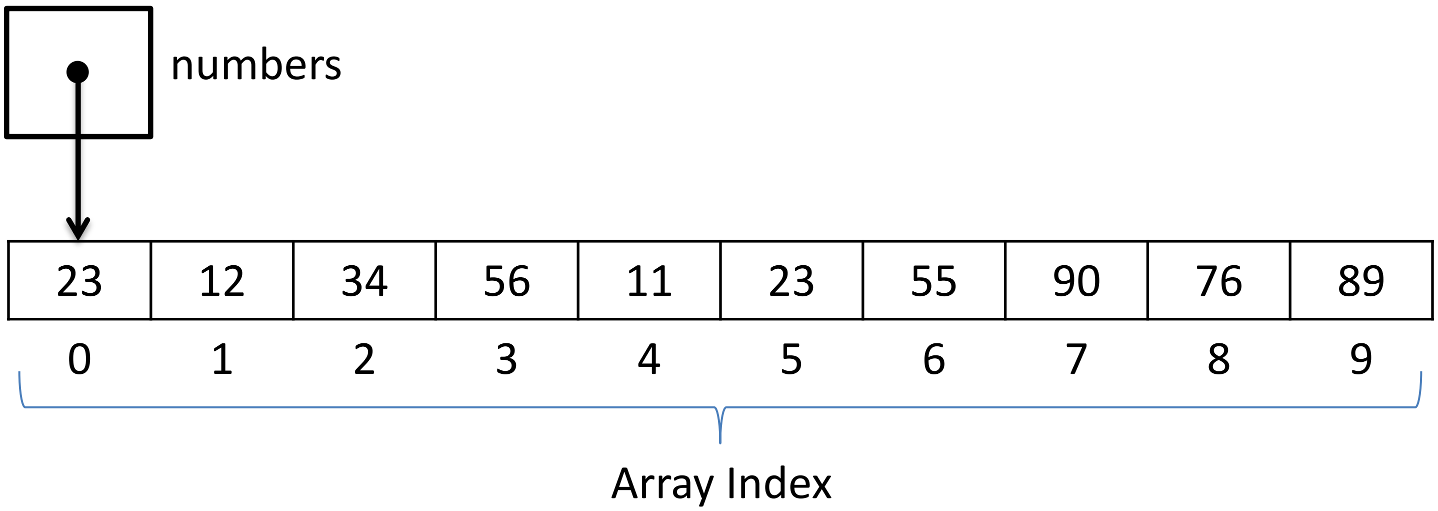
0943

Array Elements

Initializing Arrays

- An array may be initialized during declaration

```
int numbers[10] = {23, 12, 34, 56, 11,
                   23, 55, 90, 76, 89}
```



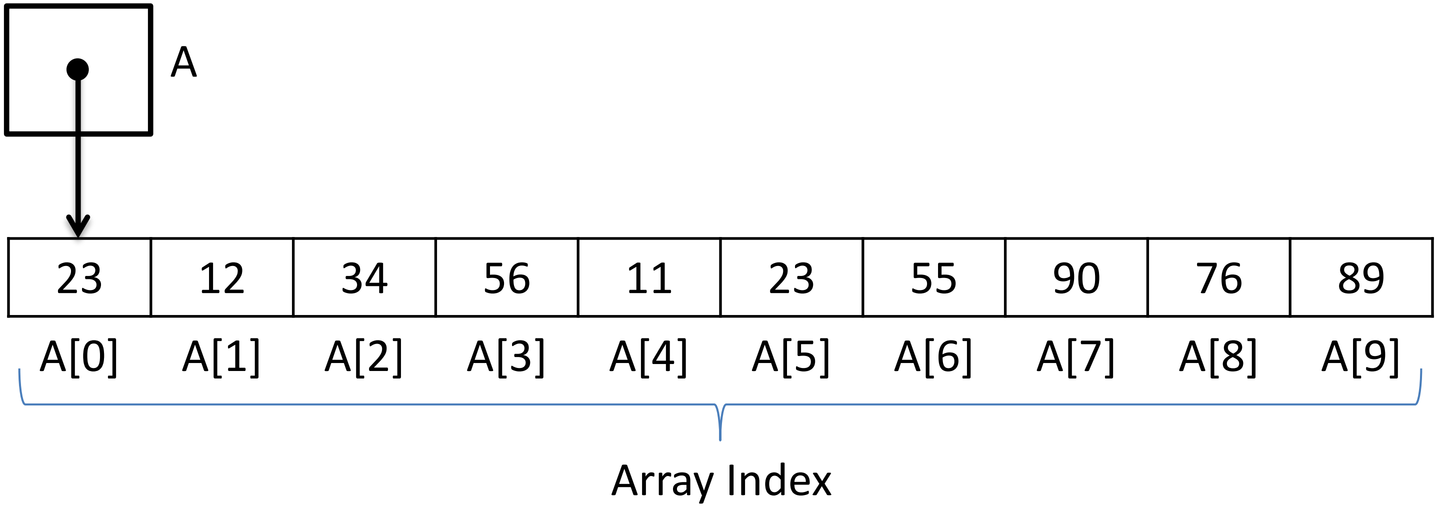
23	12	34	56	11	23	55	90	76	89
0	1	2	3	4	5	6	7	8	9

Accessing Arrays

- There are two ways of accessing array elements:
  - Using indexing
  - Using pointer arithmetic

Indexing

- Arrays are numbers successively from 0 to size-1
- The first element is index 0, the last is index size-1



23	12	34	56	11	23	55	90	76	89
A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]

Array Index

Indexing

- To access array elements through indexing:

```
<var_name>[index]
```

```
/*assign the value 10 to the 9th
element of the array numbers*/
numbers[8] = 10;
```

Notes

- When accessing array components, make sure that the index is with the bounds of the array

```
int A[10];

A[-1] = 0;       //invalid!
A[10] = 9;       //invalid!
A[4] = 3;        //valid
```

Notes

- Only integers are allowed as index

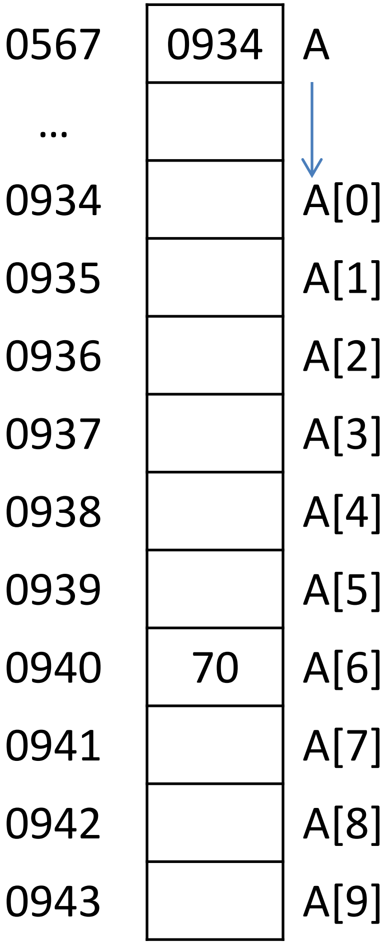
```
int A[10];
int i = 2, j = 1;

A[7.8] = 0;        //invalid!
A[j/i] = 23;       //stores 23 in A[0]
A[(i*j)%3] = 34;  //stores 34 in A[2]
```

Pointer Arithmetic

- The variable name is a constant pointer that holds the memory address of the first element of an array
- A pointer can be used to access the array elements as well
- Pointer arithmetic is done via the **indirection operator (*)**

Pointer Arithmetic



0567	0934	A
…		
0934		A[0]
0935		A[1]
0936		A[2]
0937		A[3]
0938		A[4]
0939		A[5]
0940	70	A[6]
0941		A[7]
0942		A[8]
0943		A[9]

```
A[6] is equivalent to
*(A + 6) or the 7th
array element
```

```
/*assign 70 to the 7th
element of A*/
*(A+6) = 70
```

Accessing Arrays

Elements	Indexing	Pointer Arithmetic
1st	`A[0]`	`*A or *(A+0)`
2nd	`A[1]`	`*(A+1)`
3rd	`A[2]`	`*(A+2)`
…	…	…
(n-1)th	`A[n-2]`	`*(A+(n-2))`
nth	`A[n-1]`	`*(A+(n-1))`

Accessing Arrays

Elements	Indexing	Pointer Arithmetic
$(i + 1)^{th}$ element	A[i]	*(A+i)
	element at index i	i^{th} element from the first element

Quiz (1/4)

```
A[0] = 30;            //same as *A = 30;
scanf (“%d”, &A[9]);  //same as ____(1)____
                      //or ____(2)_____
printf (“%d”, A[2]);  //same as ____(3)____
scanf (“%d”, &A[0])   //same as ____(4)____
                      //or ____(5)_____
```

Quiz (Answer)

```
A[0] = 30;            //same as *A = 30;
scanf (“%d”, &A[9]);  //same as &*(A+9)
                      //or A+9
printf (“%d”, A[2]);  //same as *(A+2)
scanf (“%d”, &A[0])   //same as &*(A)
                      //or A
```

Loops and Arrays

- For easier access of array elements, use loops together with indexing/pointer arithmetic

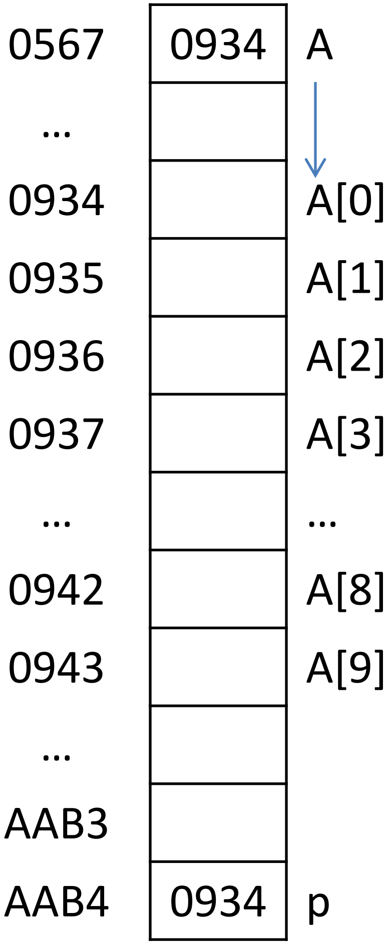
```
/*Ask 10 integers from user and assign
each to array A*/
for (i=0; i<10; i++) {
   scanf (“%d”, &A[i]);
   //scanf (“%d”, &*(A+i));
   //scanf (“%d”, A+i);
}
```

Notes

- Pointers other than the array variable name can be used to access the array elements

```
int A[5], *p;
p = &A;           //assign address of
                  //A[0] to p
*(p+2) = 24;      //assign 24 to p[2]
p[8] = 7;         //assign 7 the 9th
                  //element
```

Notes



0567	0934	A
…		
0934		A[0]
0935		A[1]
0936		A[2]
0937		A[3]
…		…
0942		A[8]
0943		A[9]
…		
AAB3		
AAB4	0934	p

p, a pointer, holds the address of the first element of array A, thus, p can be used to access the elements of A using indexing and pointer arithmetic.

Notes

- The address operator (&) can be used to obtain the address of the ith element

```
int A[10];
Int *p;

p = &A[3];   //p holds the address of
        //the 4th element of
   //array A
```

Notes

- The variable name of an array cannot hold memory locations other than the array’s first element.

```
int A[10], B[20];
int x = 8, *p;
p = &x;
A = B  //this is invalid!
B = p  //this is invalid!
A = &x //this is invalid!
```

Arrays as Parameters

- To pass arrays as actual parameters to functions, pass the array name without an index
- The address of the first element is passed to the function

```
int main () {
   int A[10];
   getInput (A);
}
```

Arrays as Parameters

- Arrays as formal parameters can be declared as
  - A pointer
  - An array with a specified size
  - An array without a specified size

Arrays as Parameters

Pointer	Array with a specified size	Array without a specified size

`int f(int *p)`
`{`
…
`}`
	`int f(int p[10])`
	`{`
	…
	`}`
		`int f(int p[])`
		`{`
		…
		`}`