ITC213: STRUCTURED PROGRAMMING

Bhaskar Shrestha

National College of Computer Studies

Tribhuvan University

Lecture 10: Arrays

Readings: Chapter 9

Introduction

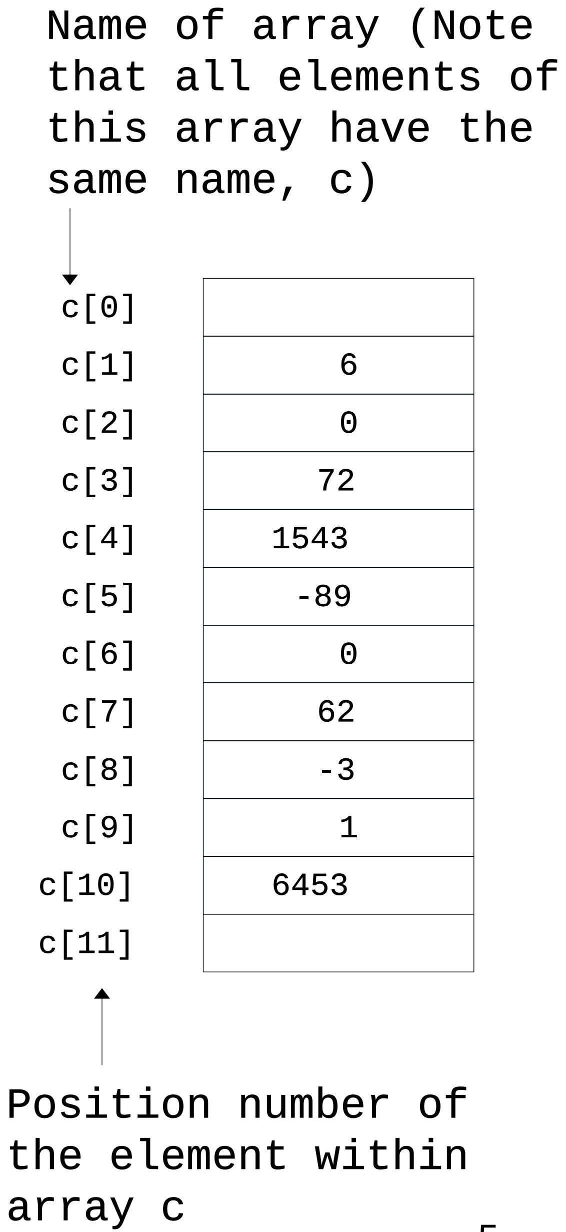
- Group of same type of variables that have same name
- Each item in the group is called an element of the array
- Each element is distinguished from another by an index
- All elements are stored contiguously in memory
- The elements of the array can be of any valid type- integers, characters, floating-point types or user-defined types

Declaring an Array

- Declared as other variables, with the array size (total no of elements) enclosed in square brackets
- Example
  - `int x[100];`
    - this creates an integer array named `x` with `100` elements
  - `char text[80];`
    - this creates a character array named `text` with `80` elements
- The size of the array specified must be a constant

Arrays

- Each array elements are distinguished with an index
- The index of first element is 0, the second element has an index of 1 and so on. The last element has an index of arraysize-1
- Example
  - `int c[12];`
    - this creates an array named c from c[0] to c[11]



Name of array (Note that all elements of this array have the same name, c)

c[0]	
c[1]	6
c[2]	0
c[3]	72
c[4]	1543
c[5]	-89
c[6]	0
c[7]	62
c[8]	-3
c[9]	1
c[10]	6453
c[11]	

Position number of the element within array c

Arrays in Memory

- The amount of storage required to hold an array is directly related to its type and size
  - *total size of array in bytes = sizeof(base type)* × *length of array*
- All arrays consist of contiguous memory locations
  - the lowest address corresponds to the first element
  - the highest address to the last element

Element	`a[0]`	`a[1]`	`a[3]`	`a[4]`	`a[5]`	`a[6]`	`a[7]`
Address	**`1000`**	**`1002`**	**`1004`**	**`1006`**	**`1008`**	**`1010`**	**`1012`**

A seven-element integer array beginning at location 1000

Manipulating Arrays

- Single operations that involve entire arrays are not permitted in C
- Each array must be manipulated on an element-by-element basis
- To access an element, specify the index of the element after array name enclosed in square brackets
  - Index must be an integral expression
- Array elements are like normal variables

```
c[0] =  3;
printf( "%d", c[ 0 ] );
```

  - Perform operations in subscript. If `x` equals `3`

```
    c[ 5 - 2 ] == c[ 3 ] == c[ x ]
```

Array Manipulation Example

```
#define NUM  100

int grade[NUM];
int i, avg, sum = 0;

printf("Input scores:\n");
for (i=0; i<NUM; i++)
   scanf("%d", &grade[i]);

for (i=0; i<NUM; i++)
   sum = sum + grade[i];
avg = sum/ NUM;
printf("Average=%d\n", avg);
```

```
/* Given the price and stock of 5 different bulbs,
calculate the total stock value */

int i, stock[5];
float price[5];
float total=0;

for (i=0; i < 5; i++)
{
  printf("Enter stock of bulb %d: ", i+1);
  scanf("%d", &stock[i]);

  printf("Enter price of bulb %d: ", i+1);
  scanf("%f", &price[i]);

  total += stock[i]*price[i];
}
printf("The total stock value is %f\n", total);
```

Array Bound Checking

- Array Bounds (index) are not verified neither at compile-time nor at run-time
- Index must be within 0 and arraysize-1
- If not others data may be overwritten

```
int a=100,b[5],c=200;
int i;
for (i=0; i < 6; i++)
   b[i] = i;
printf("a=%d, c=%d\n", a, c);
```

Initializing Arrays

- Each array element can be initialized, when an array is declared
- The initial values must appear in the order in which they will be assigned to the individual array elements, enclosed in braces and separated by commas
- Example

```
int digits[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
static float x[6] = {0, 0.25, 0, –0.50, 0, 0};
char color[3] = {'R', 'E', 'D'};
```

```
/* The wattage problem */

int i, stock[5];
int watt[5] = {15,25,40,60,100};
float price[5];
float total=0;

for (i=0; i < 5; i++)
{
  printf("Enter stock of bulb %d: ", watt[i]);
  scanf("%d", &stock[i]);

  printf("Enter price of bulb %d: ", watt[i]);
  scanf("%f", &price[i]);

  total += stock[i]*price[i];
}
printf("The total stock value is %f\n", total);
```

More on Initialization

- When a list of initializers is shorter than the number of array elements to be initialized, the remaining elements are initialized to zero

```
int digits[10] = { 3, 3, 3};
```

  - the elements digits[3] to digits[9] will have value 0
- You can use quoted strings to initialize character-arrays

```
char color[4] = "RED";
```

  - here the null character is appended by the compiler
- The array size can be omitted if you initialize the array elements

```
int digits[] = {1, 2, 3, 4, 5, 6};
```

  - the size of digits is 6

```
char color[] = "RED";
```

  - the size of color is 4

One-dimensional Arrays and Strings

- Common use for the one-dimensional array is as a character string
- A string is a null-terminated character array. (A null is zero)
- A string contains the characters that make up the string followed by a null
- When declaring a character array to hold a string, declare it to be one character longer than the largest string that it will hold

```
char str[11]
```

  - declares an array `str` that can hold a 10-character string
- When you use a quoted string constant in your program, you are also creating a null-terminated string
  - "hello there"
  - the null is automatically added by the compiler

Reading and Writing Strings

- Reading strings
  - use gets or scanf

```
char text[80];
gets(text), scanf("
  %[^\n]", text)
```

Reads characters until newline encountered

```
scanf("%s", text);
```

Reads characters until whitespace encountered

  - the null character is automatically appended
  - Can write beyond end of array, be careful

- Writing strings
  - use puts or printf

```
puts(text);
printf("%s", text);
```

Finding the length of a string

```
char text[80];
int len;
gets(text);

len = 0;
while (text[len] != '\0')
    len++;

printf("The string \"%s\" has %d
characters\n", text, len);
```

Lowercase to Uppercase Conversion

```
char text[80];
int i;
gets(text);

for (i=0; text[i] != '\0'; i++)
{
  if (text[i]>='a' && text[i]<='z')
    text[i] = text[i]-32;
}

puts(text);
```

Copying Strings

```
char str1[80], str2[80];
int i;
gets(str1);

for (i=0; str1[i] != '\0'; i++)
{
   str2[i] = str1[i];
}
str2[i] = '\0';

puts(str2);
```

Concatenating Strings

```
char s1[80], s2[80];
int i, j, len;
gets(s1);
gets(s2);

for (len=0; s1[len] != '\0'; len++)
   ;
for (i=0, j=len; s2[i] != '\0'; i++, j++)
   s1[j] = s2[i];
s1[j] = '\0';

puts(s1);
```

Searching in an Array

- Specific elements of an array can be searched in one of two ways
- Linear(Sequential) search
  - Each element is compared to the key one by one
  - Useful for small and unsorted arrays
- Binary Search
  - Can be used only on sorted arrays
  - First compares the key with the middle element of the array, if not found one-half of the array is searched in the similar way

```
/* Linear Search: Searching for key in an array
number of size max */

for (i = 0; i < max; i++)
{
 if (key == number[i])
  break;
}

if (i == max)
 printf("%d was not found\n", key);
else
 printf("\n%d was found at position %d", key, i);
```

Sorting an Array

- The process of arranging the elements such that they are according to some strict order (eg ascending/descending)
- This can be accomplished using a technique known as bubble sort
  - The rearrangement will begin by scanning the entire array for the smallest number
  - This number will then be interchanged with the first number in the array, thus placing the smallest number at the top of the list
  - Next the remaining max - 1 numbers will be scanned for the smallest, which will be exchanged with the second number
  - The remaining max - 2 numbers will then be scanned for the smallest, which will be interchanged with the third number, and so on, until the entire array has been rearranged

```
/* rearrange a list of max numbers */
for (item = 0; item < max – 1; item++)
 /*find the smallest of all remaining elements*/
 for (i = item + 1; i < max; i++)
   if (number[i] < number[item])
   {
      /* interchange the elements */
      temp = number[item];
      number[item] = number[i];
      number[i] = temp;
   }
```

Passing Arrays to Functions

- Passing Arrays
  - To pass an array argument to a function, specify the name of the array without any brackets

```
float list[100];
  .....
  avg = average(list,n);
```

  - The array name is written with an empty square bracket in the formal parameter declaration

```
float average(float x[], int n){}
```

  - Name of array is address of first element
- Passing Array Elements
  - Passed by call-by-value
  - Pass subscripted name (i.e., list[3]) to function

```
/* function prototype */
float average(float x[], int n);

int main()
{
    int n;
    float avg;
    float list[100];
    .....
    avg = average(list,n);
    .....
}

/* function definition */
float average(float x[], int n)
{
    .....
}
```

Arrays are always passed by reference

- Arrays are passed by reference
- Name of array is treated as the address of the first element in the function
  - Hence it actually becomes a pointer to the first element of the array in the function
- Function knows where the array is stored
  - Can modify original array elements passed

```c
void modify(int b, int c[]);

main() {
  int b = 2;
  int i, c[] = { 10, 20, 30 };
  modify(b,c);
  printf("b = %d\n", b);
  for (i = 0; i < 3; i++)
    printf("c[%d] = %d\n", i, c[i]);
}

void modify(int b, int c[])
{
  int i;
  b = -999;
  for (i = 0; i < 3; i++)
   c[i] = -9;
}
```

String Manipulation Library Functions

- The standard C library defines a wide range of functions that manipulate strings
  - **strcpy(s1,s2):** Copies s2 into s1
  - **strcat(s1,s2):** Concatenates s2 onto the end of s1
  - **strlen(s1):** Returns number of characters in s1 excluding the terminating null character
  - **strcmp(s1,s2);** Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s1
  - **strchr(s1,ch):** Returns a pointer to the first occurrence of the character ch in s1
  - **strstr(s1,s2):** Returns a pointer to the first occurrence of s2 in s1
- All string functions use the standard header <string.h>

```
char name[40], first[40];

printf("Enter a name: ");
gets(name);

strcpy(first, name);
while (strcmp(name,"END") != 0) {
  if (strcmp(first,name) > 0))
     strcpy(first, name);
  printf("Enter a name: END to stop");
  gets(name);
}

printf("The first is %s\n", first);
```

Multi-dimensional Arrays

- Recall: An array is a sequence of data items that are of the same type, that can be indexed, and that are stored contiguously
- Each element of an array is like a single item of a particular type
- But an array itself is an item of a particular type
  - So, an array element could be another array
- An “array-of-arrays” is called “multi-dimensional” arrays whose elements are themselves arrays
  - No of subscript determines the dimension of the array

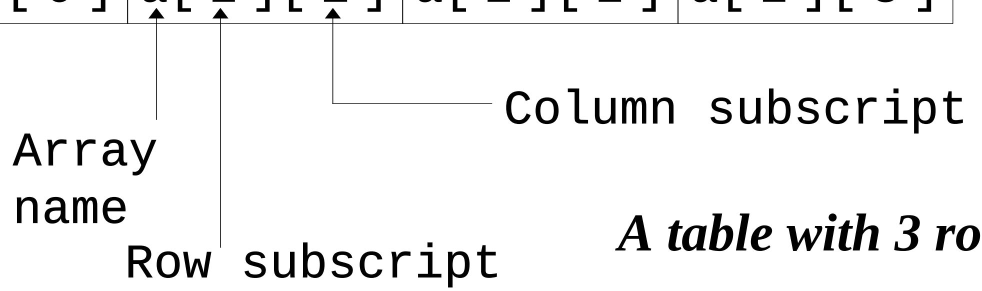
Two-dimensional Arrays

- A two-dimensional array is an array of one-dimensional arrays
- Example: `int a[3][4];`
  An array of 3 elements, in which every element is an array of 4 ints
- Accessing Elements
  - `a[1]`
    This gives the second element, i.e., second array (address of first element of second array)
  - `a[1][2]`
    This gives the third integer within the second array

Two-dimensional Arrays

- Think, two-dimensional arrays as tables/matrices arranged in rows and columns
- Use first subscript to specify row no and the second subscript to specify column no

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]



Column subscript

Array name

Row subscript

A table with 3 rows and 4 columns

```
int a[3][4];
int i, j;

for (i = 0; i < 3; ++i)
   for (j = 0; j < 4; ++j)
      a[i][j] = i+j;

for (i =0; i < 3; ++i)
{
   for (j = 0; j < 4; ++j)
      printf("a[%d][%d] = %d  ", i, j, a[i][j]);
   printf("\n");
}

printf("%d\n", a[2][1]/2);
printf("%d\n", a[1][1] * (a[0][0]+2));
printf("%d\n", a[3][1]/2); /* ERROR: ? */
```

Initialization

- List the values separated by commas and enclosed in braces
  - `int a[2][3] = { 1, 2, 3, 4, 5, 6};`
  - The values will be assigned in the order they appear
- Initializers can be grouped with braces
  - `int a[2][3] = { {1, 2, 3}, {4, 5, 6}};`

1	2	3
4	5	6

- If not enough, unspecified elements set to zero
  - `int a[2][3] = { {1, 2}, {3, 4}};`

1	2	0
3	4	0

- You can leave the size for first subscript
  - `int a[][3] = { {1, 2}, {3, 4}};`

Passing Multidimensional Arrays to Function

- Specify the array variable name, while passing it to a function
  - only the address of the first element is actually passed
- The parameter receiving the array must define the size of all dimension, except the first one
- Any changes to array elements within the function affects the “original” array elements

```
int a[3][4];
func(a);
```

Function Call

```
void func(int x[][4])
{
}
```

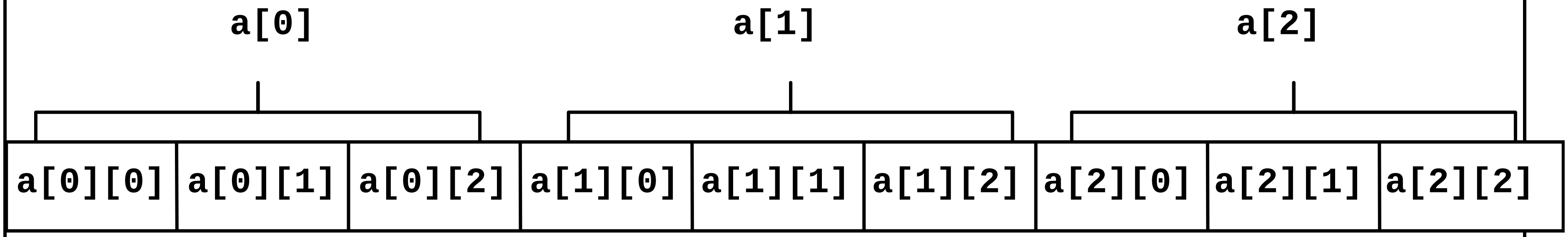
Multidimensional array in parameter

```c
#define MAXROWS 10
#define MAXCOLS 20

void ReadTable(int t[][MAXCOLS], int r, int c);
void PrintTable(int t[][MAXCOLS], int r, int c);
int SumOfOddElements(int t[][MAXCOLS], int r, int c);
int SumOfEvenElements(int t[][MAXCOLS], int r, int c);

main()
{
     int table[MAXROWS][MAXCOLS];
     int nrows, ncols;
     int oddsum, evensum;

     printf("Enter no of rows and columns: ");
     scanf("%d %d", &nrows, &ncols);

     ReadTable(table, nrows, ncols);

     oddsum = SumOfOddElements(table, nrows, ncols);
     evensum = SumOfEvenElements(table, nrows, ncols);
     PrintTable(table, nrows, ncols);
     printf("Odd sum = %d, Even sum = %d\n", oddsum, 
evensum);

}

void ReadTable(int t[][MAXCOLS], int r, int c)
{
     int i, j;
     for (i = 0; i < r; i++)
     {
          printf("Enter elements for row %d\n", i+1);
          for (j = 0; j < c; j++)
          {
               printf("Column %d: ", j+1);
               scanf("%d", &t[i][j]);
          }
     }
}
```

```
void PrintTable(int t[][MAXCOLS], int r, int c)
{
     int i, j;
     for (i = 0; i < r; i++)
     {
          for (j = 0; j < c; j++)
               printf("%5d", t[i][j]);
          printf("\n");
     }
}

int SumOfOddElements(int t[][MAXCOLS], int r, int c)
{
     int i, j;
     int sum = 0;
     for (i = 0; i < r; i++)
          for (j = 0; j < c; j++)
               if (t[i][j]%2 != 0)
                    sum += t[i][j];
     return sum;
}
```

Multidimensional Arrays in Memory

- Each array within a multidimensional array stored sequentially in memory as with one-dimensional array
- For two-dimensional array, all elements in first row is stored, then the elements of second row and so on



a[0]			a[1]			a[2]		
a[0][0]	a[0][1]	a[0][2]	a[1][0]	a[1][1]	a[1][2]	a[2][0]	a[2][1]	a[2][2]

Array of Strings

- You can create array of strings using a two-dimensional character array

  ```
  char months[12][10];
  ```

  – Left dimension determines the number of strings, and right dimension specifies the maximum length of each string

  – Now you can use the array `months` to store 12 strings each of which can have a maximum of 10 characters (including the null)

  – To access an individual string, you specify only the left subscript

  ```
  puts(months[2]);
  ```

  prints the third month

Example

```
char months[12][10] =
{
"January",
"February",
"March",
"April",
"May",
"June",
"July",
"August",
"September",
"October",
"November",
"December“
};
printf("%s\n", months[5]);
```

months[0]	J	a	n	u	a	r	y	\0		
months[1]	F	e	b	r	u	a	r	y	\0	
months[2]	M	a	r	c	h	\0				
months[3]	A	p	r	i	l	\0				
months[4]	M	a	y	\0						
months[5]	J	u	n	e	\0					
months[6]	J	u	l	y	\0					
months[7]	A	u	g	u	s	t	\0			
months[8]	S	e	p	t	e	m	b	e	r	\0
months[9]	O	c	t	o	b	e	r	\0		
months[10]	N	o	v	e	m	b	e	r	\0	
months[11]	D	e	c	e	m	b	e	r	\0	