5/13/2019



# **Chapter7: Arrays**

#### **Computer Science Department**

#### **Arrays - Introduction**

- Simple data types use a single memory cell to store a variable.
- Sometimes, we need to *group data items* together in main memory than to allocate an individual memory cell for each variable.
- Example: A program that processes exam scores for a class.
- Here, it would be easier to write if all the scores were <u>stored in one area</u> of memory and were able to be <u>accessed as a group</u>

### **Arrays - definition**

Array is a collection of data items of the same type.

Array element is a data item that is part of an array.





- Array
  - Group of consecutive memory locations
  - Same name and type
- To refer to an element, specify
  - Array name
  - Position number
- Format:

arrayname[ position number ]

- First element at position 0
- n element array named c:
  - c[ 0 ], c[ 1 ]...c[ n 1 ]



Position number of the element within array c

## **Declaring Arrays**

• When declaring arrays, specify

arrayType arrayName[numberOfElements];

e.g. int c[ 10 ];
 float myArray[ 100 ];

Declaring multiple arrays of same type

- Format similar to regular variables

e.g. int b[ 100 ], x[ 27 ];

### **Declaring Arrays**



### Arrays

Array elements are like normal variables

```
c[ 0 ] = 3;
printf( "%d", c[ 0 ] );
c[1]= c[0]+c[2]
c[3]= c[2]+5
```

Perform operations in subscript (index).

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

#### **Example: Operations on Arrays**

double x[8];

Array x

x(0) x(1) x(2) x(3) x(4) x(5) x(6) x(7) 16.0 12.0 6.0 8.0 2.5 12.0 14.0 -54.5

#### Statements that manipulate the value of x

Statement	Ex	planati	on							
printf("%.1f", x[0]);	Displays the value of $x[0]$ , which is $16.0$ .									
x[3] = 25.0;	St	ores the	value 2	5.0 in :	x[3].					
<pre>sum = x[0] + x[1];</pre>	Stores the sum of $x[0]$ and $x[1]$ , which is 28.0 in the variable sum.									
sum += x[2];	Ac	dds x [ 2	] to su	m. The r	new sur	n is 34.	0.			
x[3] += 1.0;	Ac	dds 1.0	to x [ 3	]. The r	new x [ 3	3] is 26	.0.			
x[2] = x[0] + x[1];	Stores the sum of $x[0]$ and $x[1]$ in $x[2]$ . The new $x[2]$ is $28.0$ .									
The array	x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]		
x is now:	16.0	12.0	28.0	26.0	2.5	12.0	14.0	-54.5		

### **Examples Using Arrays**

#### Initializers

```
int n[5] = { 1, 2, 3, 4, 5 };
char alphabet[5] = { `A', 'B', 'C', 'D', 'E' };
```

All elements 0

```
int n[5] = \{0\}
```

If size omitted, initializers determine it

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

5 initializers, therefore 5 element array

### **Example Array**

```
/* prints the days for each month */
#include <stdio.h>
#define MONTHS 12
int main(void)
{
int days[MONTHS] = {31,28,31,30,31,30,31,30,31,30,31,30,31}; Month 11 has 30 days.
int index;
for (index = 0; index < MONTHS; index++)</pre>
  printf("Month %d has %2d days.\n", index +1,days[index]);
return 0;
}
```

Month 1 has 31 days. Month 2 has 28 days. Month 3 has 31 days. Month 4 has 30 days. Month 5 has 31 days. Month 6 has 30 days. Month 7 has 31 days. Month 8 has 31 days. Month 9 has 30 days. Month 10 has 31 days. Month 12 has 31 days.

# Example: Fill and Print Array

```
#include <stdio.h>
int main ()
                                                                Element[0] = 1
                                                                Element[1] = 2
  int n[ 10 ]; // n is an array of 10 integers
  int i,j;
                                                                Element[2] = 3
                                                                Element[3] = 4
   // initialize elements of array
                                    0 (Fill Array)
                                                                Element[4] = 5
  for ( i = 0; i < 10; i++ )
  -{
                                                                Element[5] = 6
     n[ i ] = i + 1; /* set element at location i to i + 1 */
                                                                Element[6] = 7
                                                                Element[7] = 8
   // output each array element's value (Print Array)
                                                                Element[8] = 9
  for (j = 0; j < 10; j++ )</pre>
                                                                Element[9] = 10
     printf("Element[%d] = %d\n", j, n[j] );
  }
  return 0;
}
```

### **Example: Fill and Print Array**

```
#include <stdio.h>
#define size 5 // array size= 5
int main ()
{
    int n[ size ]; // n is an array of 5 integers
    int i, j;
    // initialize elements of array n (Fill Array)
    for ( i = 0; i < size; i++ )
    {
        scanf ("%d", &n[ i ]);
    }
    // output each array element's value (Print Array)
    for (j = 0; j < size; j++ )
    {
        printf("Element[%d] = %d\n", j, n[j] );
    }
    return 0;
</pre>
```

Input:		
12345		
Output:		
Element[0]	=	1
Element[1]	=	2
Element[2]	=	3
Element[3]	=	4
Element[4]	=	5

#### **Multi- Dimensional Arrays**

- Multidimensional arrays are derived from the basic or built-in data types of the C language.
- Two-dimensional arrays are understood as rows and columns:
  - two-dimensional tables,
  - parallel vectors,
  - two-dimensional matrices.
- Mostly Two-dimensional array are used in Multi-dimensional array







STUDENTS-HUB.com

Uploaded By: anonymous

### What is a Two-dimensional array?



#### How to initialize a Two-Dimensional array?

- Initialized directly in the declaration statement
  - int b[2][3] = {51, 52, 53, 54, 55, 56};
  - b[0][0] = 51 b[0][1] = 52 b[0][2] = 53
- Use braces to separate rows in 2-D arrays.

Implicitly declares the number of rows to be 4.

### 2-Dimenstional Array

Create array elements by telling how many ROWS and COLUMNS

Example:

int grades[5][3];

grades is a *two-dimensional array*, with 5 rows and 3 columns. One row for each student. One column for each test.

### Declare & Initialize

#### Example:

```
int grades[5][3] =
  { { 78, 83, 82 },
     { 90, 88, 94 },
     { 71, 73, 78 },
     { 97, 96, 95 },
     { 89, 93, 90 } };
```

A Two-D Array is an *array of arrays*. Each row is itself a One-D array.

### Row, Column Indices

	0	1	2
0	78	83	82
1	90	88	94
2	71	73	78
<u>3</u>	97	96	<u>95</u>
4	89	93	90

Give both the ROW and COLUMN indices to pick out an individual element.

The fourth student's third test score is at ROW 3, COLUMN 2

### Example

int	a[2]	[4]	;
-----	------	-----	---

a[1][0]=9;

a[0][3]=5;

	0	1	2	3
0		14		5
1	9			

a[0][1]=a[0][3]+ a[1][0];

### **Representation of Arrays**

A computer's memory is a *one dimensional array of cells*.

How is a 2-D array stored?

<u>Answer</u>: In C, the array rows are stored sequentially: row 0, 1, 2, ...

Represe	enta	ati	on	0	t A	\rr	ays
score	hw 0	1	2	3	4	5	
student 0	13	15	25	25	?	?	
student 1	12	12	25	20	?	?	
student 2	5	17	25	24	?	?	
student 3	15	19	25	13	?	?	
student 4	2	0	25	25	?	?	
student 5	25	22	24	21	?	?	
student 6	8	4	25	12	?	?	
student 7	?	?	?	?	?	?	

۲-23



R-26

### **Representation of Arrays**

score	hw 0		2	3	4	5							
student 0	13	15	25	25	?	?							
student 1	12	12	25	20	?	?							
student 2	5	17	25	24	?	?							
student 3	15	19	25	13	?	?							
student 4	2	0	25	25	?	?							
student 5	25	22	24	21	?	?							
student 6	8	4	25	12	?	?							
student 7	?	?	?	?	?	?							
1315 25 25 2	2 1	2 1	2 2!	5 20	2	2	5	17	25	24	2	2	1
				20		•	<u> </u>		20	27		•	
student 0	•												





Uploaded By: anonymous

### Example : Fill Array

```
What are the elements of the array table?
int table[3][4];
int x = 1;
for (row = 0; row < 3; row++)
    for (col = 0; col < 4; col++)
    {
       table[row][col] = x;
       x++;
    } //for col</pre>
```

0-30

### Array Elements as Parameters

**Individual** array elements can be used as parameters, *just like other simple variables*. Examples:

printf( "Last two are %f, %f", rain[5], rain[6] );

draw\_house( color[i], x[i], y[i], windows[i] );

scanf( "%lf", &rain[0] ) ;

swap( &rain[i], &rain[i+1] );

### Whole Arrays as Parameters

Array parameters (entire arrays) work differently: An array is never copied (<u>no call by value</u>) The array name is *always treated as a pointer parameter* The & and \* operators are not used

Programming issue: in C, arrays do not contain information about their size, so the size often needs to be passed as an additional parameter.

0-32

### Array Parameter Example

```
#define ARRAY_SIZE 200
double average ( int a[ARRAY_SIZE ) {
    int i, total = 0;
    for ( i = 0 ; i < ARRAY_SIZE ; i = i + 1 )
        total = total + a[i] ;
    return ((double) total / (double) ARRAY_SIZE);
}
int x[ARRAY_SIZE];
...
x_avg = average ( x ) ;</pre>
```

### **General Vector Sum**

Usually the size is omitted in an array parameter declaration.

int x[3] = {1,2,3}, y[3] = {4,5,6}, z[3]; VectorSum(x,y,z,3);

# Write a program that adds two 2x2 arrays and stores the sum in third array.

```
int i,j;
#include <stdio.h>
                                                       for (i=0;i<rows;i++) {</pre>
#include <stdlib.h>
                                                          for (j=0;j<cols;j++)</pre>
                                                                                    {
#define rows 2
                                                          printf("array[%d][%d]: ",i,j);
#define cols 2
                                                           scanf("%d",&array[i][j]);
void Fill(int [][cols]);
void Sum(int [][cols],int [][cols],int [][cols]);
                                                         } }
                                                       printf ("\n\n\n");
void Print(int [][cols]);
int main()
                                 void Sum(int array_a[][cols],int array_b[][cols],int result[][cols])
{
                                 {
  int array_a[rows][cols];
                                    int i,j;
  int array b[rows][cols];
                                    for (i=0;i<rows;i++) {</pre>
  int result[rows][cols];
                                      for (j=0;j<cols;j++)</pre>
                                                                 {
  printf ("array 1: \n");
                                         result[i][j]=array_a[i][j]+array_b[i][j];
  Fill(array_a);
                                      } } }
  printf ("array 2: \n");
  Fill(array_b);
                                                     void Print(int array[][cols]) {
                                                      int i,j;
  Sum(array_a,array_b,result);
                                                      for (i=0;i<rows;i++) {</pre>
  printf ("result: \n");
                                                         printf ("\n");
  Print(result);
                                                         for (j=0;j<cols;j++)
                                                         printf("%d ",array[i][j]);
  return 0:
                                                        3 3
}
                                                      printf ("\n\n\n");
```

3

#### Fill and print array using function & reverse

```
#include <stdio.h>
                                                 void printArray (int myArray[],int s)
#define size 5 // array size= 5
                                                 {
void fillArray (int[],int);
                                                    int i:
void printArray (int[],int);
                                                    for (i=0;i<s;i++){
void printArrayInreverse(int[],int);
                                                       printf ("myArray[%d]= ",i);
int main () {
                                                       printf("%d",myArray[i]);
  int n[ size ];
                                                       printf("\n");
  printf("Fill Array\n-----\n");
  fillArray(n,size);
  printf("Print Array\n-----\n");
                                                    }
  printArray(n,size);
  printf("\nReverse Array\n-----\n");
                                                 }
  printArrayInreverse(n,size);
  return 0;
                                              void printArrayInreverse(int myArray[],int s)
                                              { int i;
void fillArray (int myArray[],int s) {
                                                for (i=s-1;i>=0;i--){
  int i;
                                                   printf ("myArray[%d]= ",i);
  for (i=0;i<s;i++)
                                                  printf("%d",myArray[i]);
  {
                                                   printf("\n");
    printf ("myArray[%d]= ",i);
                                                }
    scanf("%d",&myArray[i]);
                                              }
    printf("\n");
 }
```

### **Example: Finding the Maximum**

```
#include <stdio.h>
#define size 5
int main()
{
    int i,max;
    int list[size];
    //initialize the array
    for (i=0;i<size;i++)
        scanf("%d",&list[i]);
    //find maximum value
    max=list[0];
    for (i=1;i<size;i++)
        if (max<list[i])
        max=list[i];
    printf("Maximum value:%d",max);
    return 0;
}</pre>
```

#### Strings

A string is a sequence of characters treated as a group We have already used some string literals: "filename" "output string" Strings are important in many programming contexts: names other objects (numbers, identifiers, etc.)

### Strings in C

No explicit type, instead strings are <u>maintained</u> as arrays of characters Representing strings in C stored in arrays of characters array can be of any length end of string is indicated by a *delimiter*, the zero character '\0'



#### **String Literals**

String literal values are represented by sequences of characters between double quotes (")
Examples
"" - empty string
"hello"
"a" versus 'a'
'a' is a single character value (stored in 1 byte) as the ASCII value for a
"a" is an array with two characters, the first is a, the second is the character value \0

#### **Referring to String Literals**

String literal is an array, can refer to a single character from the literal as a character Example:

printf("%c","hello"[1]); outputs the character 'e'

During compilation, C creates space for each string literal (# of characters in the literal + 1)

### **Duplicate String Literals**

Each string literal in a C program is stored at a *different location* 

So even if the string literals contain the same string, <u>they are not equal</u> (in the == sense) Example: char string1[6] = "hello";

char string2[6] = "hello"; but *string1 does not equal string2* (they are stored at different locations)

#### **String Variables – Declaration**

```
Allocate an array of a size large enough to hold
the string (plus 1 extra value for the delimiter)
Examples (with initialization):
char str1[6] = "Hello";
char str2[] = "Hello";
char str3 = "Hello";
char str4[6] = {'H','e','I','I','o','\0'};
Note, each variable is considered a constant in
that the space it is connected to cannot be
<u>changed</u>
str1 = str2; /* not allowable, but we can copy the contents
of str2 to str1 (more later) */
```

### **Changing String Variables**

Cannot change **space** string variables connected to, but *can use pointer variables that can be changed* 

#### Example:

```
char *str1 = "hello"; /* str1 unchangeable */
char *str2 = "goodbye"; /* str2 unchangeable */
char *str3; /* Not tied to space */
str3 = str1; /* str3 points to same space s1 connected to */
str3 = str2;
```

### **Changing String Variables (cont)**

Can change parts of a string variable char str1[6] = "hello"; str1[0] = 'y'; /\* str1 is now "yello" \*/ str1[4] = '\0'; /\* str1 is now "yell" \*/

**Important**: to retain delimiter (replacing str1[5] in the original string with something other than '\0' *makes a string that does not end*) Have to stay within limits of array

### Linear and Binary Search

### **Linear Search**

#### Problem

Given a list of N values, determine whether a given value X occurs in the list. **Example**: consider the problem of determining whether the value 55 occurs in:

1 2 3 4 5 6 7 8 17 31 9 73 55 19 12 7

#### ALGORITHM

- 1. Assume the target has not been found.
- 2. Start with the initial array element.
- 3. repeat while the target is not found and there are more array elements
  - 4. if the current element matches the target
    - 5. Set a flag to indicate that the target has been found.

else

- 6. Advance to the next array element.
- 7. if the target was found
  - 8. Return the target index as the search result.

else

9. Return -1 as the search result.

#### Code:

```
search(const int arr[], /* input - array to search
                                                                          */
      int target, /* input - value searched for
                                                                          */
      int
               n)
                       /* input - number of elements to search
                                                                          */
{
     int i,
        found = 0,
                       /* whether or not target has been found
                                                                         */
                       /* index where target found or NOT FOUND
                                                                          */
         where:
                                                                          */
     /* Compares each element to target
     i = 0;
     while (!found && i < n) {
         if (arr[i] == target)
              found = 1;
         else
               ++i;
     }
     /* Returns index of element matching target or NOT_FOUND
                                                                         */
     if (found)
           where = i;
                                       Calling the search function:
     else
           where = NOT FOUND;
                                       index = search(ids, 4902, SIZE);
     return (where);
```

#### Linear Search – Example 2

```
#include <stdio.h>
#define size 7
int main() {
int myArray[size]={209,99,887,01234,987,54,66};
 int target; // input - value searched for
 int location; // index of the target
                                              if (found==1)
 int found = 0;
                                                  printf("Matching target,
  int i=0:
                                             location is %d\n",location);
  printf("please enter a target: ");
                                                else
  scanf("%d",&target);
                                                   printf("Not found\n");
  while (i<size) {
                                               return 0;
       if (target==myArray[i]) {
                                             } // main
         location=i; //update location
         found=1; //Matching target
         break:
        } // end if
       i++;
   } // end while
```

## **Sorting: Selection Sort**

#### **Example:** Sorting in descending order

void Sort(int array[]) #include <stdio.h> { #include <stdlib.h> #define Size 3 int i,j; void Sort (int []); int temp; int main() { { int i; int array[Size]; printf("Enter array size %d\n",Size); for(i=0;i<Size;i++)</pre> scanf("%d",&array[i]); Sort (array); printf("array after sorted :"); for(i=0;i<Size;i++)</pre> } printf("%d ",array[i]); } printf("\n"); return 0; }

for(i=0;i<Size-1;i++)</pre> for (j=i+1;j<Size;j++) { if (array[i]<array[j]) { temp=array[j]; array[j]=array[i]; array[i]=temp; } // if statement } // inner loop

// outer loop

// Sort function

Enter array of integers with size 3 345 array after sorted :5 4 3