Arrays in C

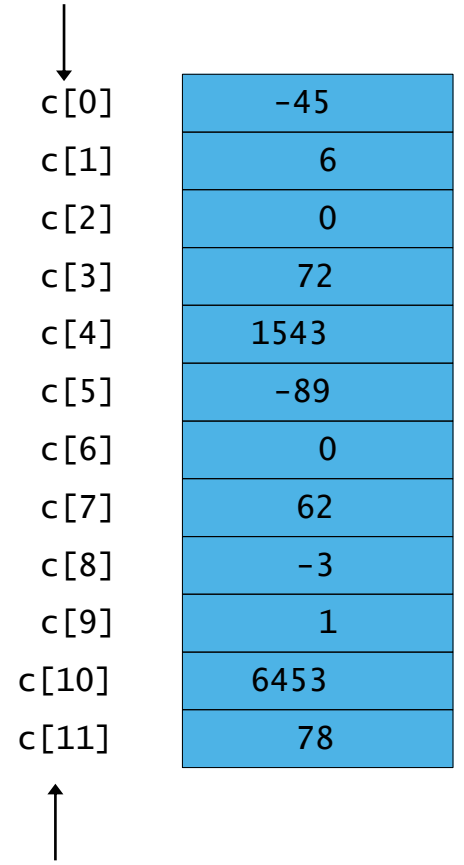
By: Atiya Jokhio

Arrays

- *An array is a collection of variables of a certain type, placed contiguously in memory.*
- 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]

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



Position number of the element within array c

Arrays (Cont'd)

- 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 ]
```

- When defining arrays, specify
  - Name
  - Type of array
  - Number of elements

```
arrayType arrayName[ numberOfElements ];
```

  - Examples:

```
int c[ 10 ];
float myArray[ 3284 ];
```

- Defining multiple arrays of same type
  - Format similar to regular variables
  - Example:

```
int b[ 100 ], x[ 27 ];
```

Arrays (Cont'd)

- Initializers

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

  - If not enough initializers, rightmost elements become 0

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

    - All elements 0

- If size omitted, initializers determine it

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

  - 5 initializers, therefore 5 element array

Arrays (Cont'd)

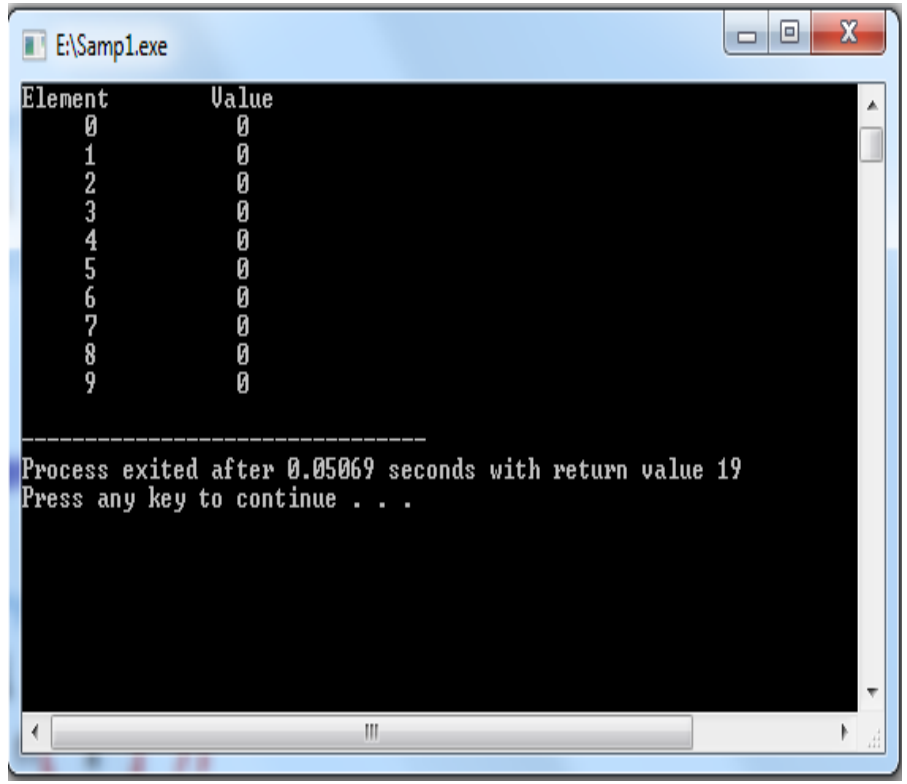
```
#include <stdio.h>

// function main begins program execution
 int main( void )
 {
    int n[ 10 ]; // n is an array of 10 integers
    int i;

    // initialize elements of array n to 0
for ( i = 0; i < 10; ++i ) {
n[ i ]= 0; // set element at location i to 0
} // end for

printf( "%s%13s\n", "Element", "Value" );

// output contents of array n in tabular format
for ( i = 0; i < 10; ++i ) {
printf( "%6d%12d\n", i, n[ i ] );
} // end for
}
```



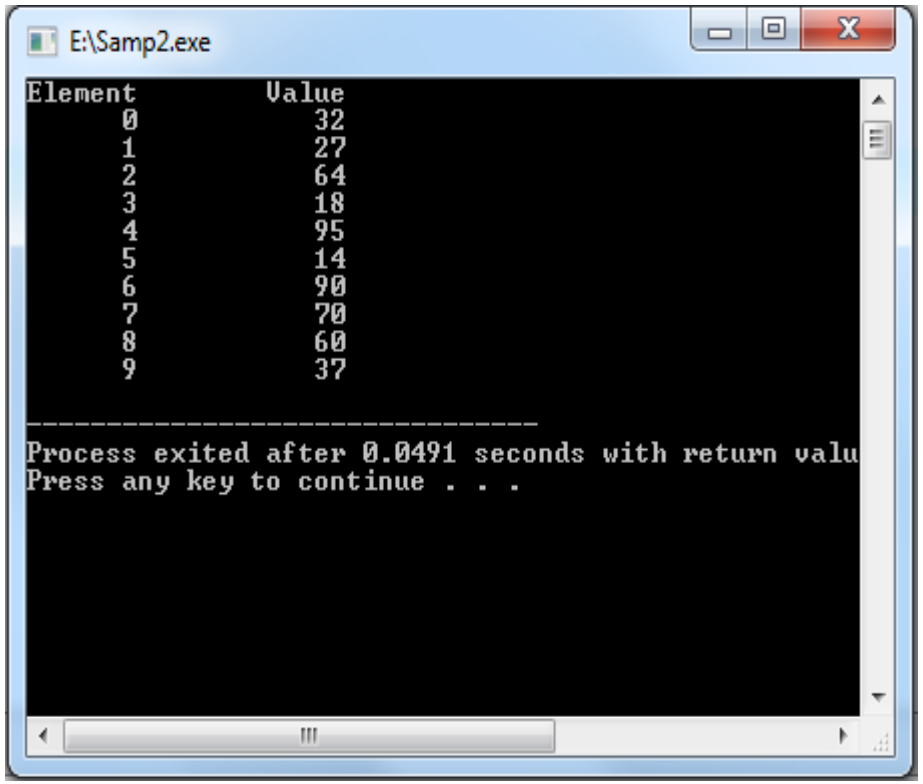
Arrays (Cont'd)

```c
#include <stdio.h>

// function main begins program execution
 int main( void )
 {
    int n[ 10 ]= { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
    int i;


    printf( "%s%13s\n", "Element", "Value" );

// output contents of array in tabular format
     for ( i = 0; i < 10; ++i ) {
     printf( "%7u%13d\n", i, n[ i ] );
    } // end for
    } // end main
```



Arrays (Cont’d)

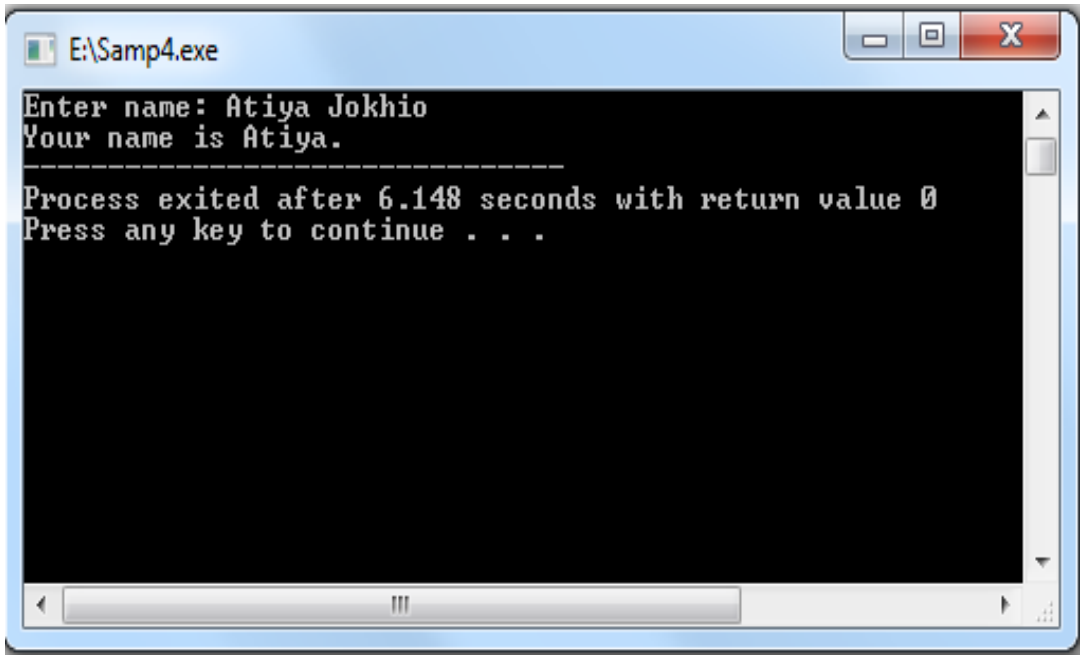
Characters Array or Strings
A string constant is a one-dimensional array of characters terminated by a null (‘\0’).
The terminator define end of string.
For example,
char name[5] = { ‘F’, ‘A’, ‘S’, ‘T’, ‘\0’ };

- Input function scanf() can be used with %s format specifier to read a string input from the terminal. But there is one problem with scanf() function, it terminates its input on the first white space it encounters. Therefore if you try to read an input string "Hello World" using scanf() function, it will only read Hello and terminate after encountering white spaces.

Arrays (Cont'd)

Characters Array or Strings

```c
#include <stdio.h>
int main()
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```



Arrays (Cont'd)

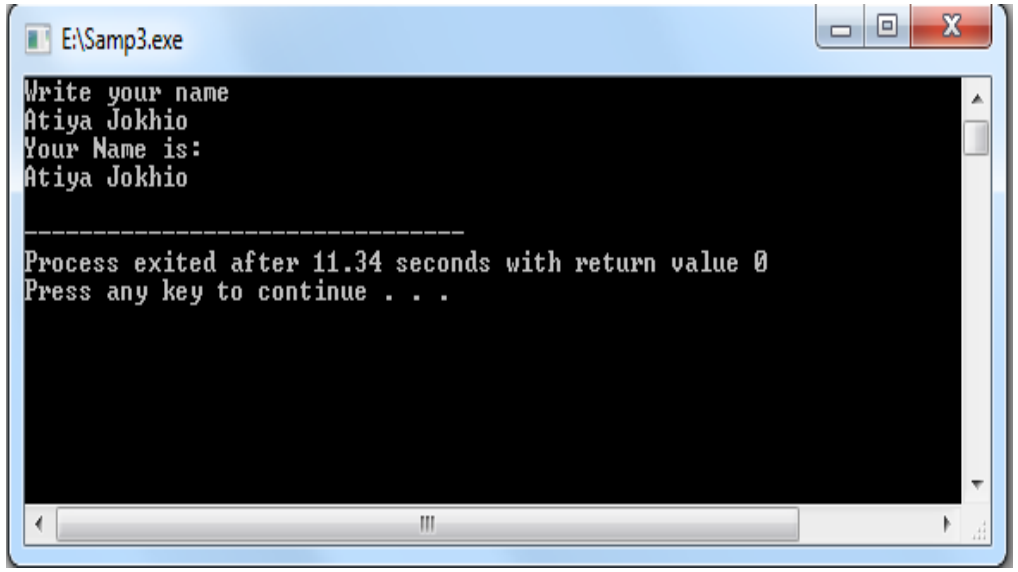
The String I/O Function gets() & puts()

Sanf() and printf() is not versatile for string I/O we can use gets() and puts() function from stdio library.

For Example:

```c
#include <stdio.h>

int main( void )
{
    char name[20];
    printf("Write your name\n");
    gets( name);
    printf("Your Name is:\n");
    puts(name);
}
```



Preprocessor Directives

- The **C preprocessor executes *before a program is compiled.***
- Normal program statements are instructions to the microprocessor; preprocessor directives are instructions to the compiler.
- The preprocessor more or less provides its own language which can be a very powerful tool to the programmer. Recall that all preprocessor directives or commands begin with a #.

Preprocessor Directives

- Use of the preprocessor is advantageous since it makes:
  - programs easier to develop,
  - easier to read,
  - easier to modify
  - C code more transportable between different machine architectures.
- Here we'll examine two of the most common preprocessor directives, #define and #include.

The #define Directive(Symbolic Constant)

The simplest use for the define directive is to assign names to constants. e.g.

```
#define PI 3.14159 //macro definition
float area (float);
  void main()
  {
        float radius;
        printf(“Enter radius of sphere: “);
        scanf(“%f”,&radius);
        printf(“Area of sphere is %.2f”, area(radius) );
  }
  float area( float rad)
  {     return(4 * PI * rad * rad );     }
```

Case Study: Computing Mean, Median and Mode Using Arrays

- Mean – average
- Median – number in middle of sorted list
  - 1, 2, 3, 4, 5
  - 3 is the median
- Mode – number that occurs most often
  - 1, 1, 1, 2, 3, 3, 4, 5
  - 1 is the mode

Sorting Arrays

- **Sorting data**
  - Important computing application
  - Virtually every organization must sort some data
- **Bubble sort (sinking sort)**
  - Several passes through the array
  - Successive pairs of elements are compared
    - If increasing order (or identical), no change
    - If decreasing order, elements exchanged
  - Repeat
- **Example:**
  - original: 3 4 2 6 7
  - pass 1: 3 2 4 6 7
  - pass 2: 2 3 4 6 7
  - Small elements "bubble" to the top

Searching Arrays: Linear Searching

- Search an array for a *key value*
- Linear search
  - Simple
  - Compare each element of array with key value
  - Useful for small and unsorted arrays

Searching Arrays:Binary Searching

- Binary search
  - For sorted arrays
  - Compares `middle` element with `key`
    - If equal, match found
    - If `key < middle`, looks in first half of array
    - If `key > middle`, looks in last half
    - Repeat
  - Very fast; at most n steps, where 2^n > number of elements
    - 30 element array takes at most 5 steps
      - 2^5 > 30 so at most 5 steps

TASK to understand

- Linear Search in array
- Binary Search in array
- Ascending elements
- Descending elements
- Bubble sort in array
- Minimum element in array
- Maximum element in array

Using Arrays to Summarize Survey Results

- Forty students were asked to rate the quality of the food in the student cafeteria on a scale of 1 to 10 (1 means awful and 10 means excellent). Place the 40 responses in an integer array and summarize the results of the poll.

Using Arrays to Summarize Survey Results

```c
// Analyzing a student poll.
#include <stdio.h>
#define RESPONSES_SIZE 40 // define array sizes
#define FREQUENCY_SIZE 11

// function main begins program execution
int main( void )
{
   size_t answer; // counter to loop through 40 responses
   size_t rating; // counter to loop through frequencies 1-10

   // initialize frequency counters to 0
   int frequency[ FREQUENCY_SIZE ] = { 0 };

   // place the survey responses in the responses array
   int responses[ RESPONSES_SIZE ] = { 1, 2, 6, 4, 8, 5, 9, 7, 8, 10,
         1, 6, 3, 8, 6, 10, 3, 8, 2, 7, 6, 5, 7, 6, 8, 6, 7, 5, 6, 6,
         5, 6, 7, 5, 6, 4, 8, 6, 8, 10 };

   // for each answer, select value of an element of array responses
   // and use that value as subscript in array frequency to
   // determine element to increment
   for ( answer = 0; answer < RESPONSES_SIZE; ++answer ) {
      ++frequency[ responses [ answer ] ];
   } // end for

   // display results
   printf( "%s%17s\n", "Rating", "Frequency" );

   // output the frequencies in a tabular format
   for ( rating = 1; rating < FREQUENCY_SIZE; ++rating ) {
      printf( "%6d%17d\n", rating, frequency[ rating ] );
   } // end for
} // end main
```

Rating	Frequency
1	2
2	2
3	2
4	2
5	5
6	11
7	5
8	7
9	1
10	3

Passing array to function :

- Array can be passed to function by two ways :
  - **Pass Array element by element**
  - **Pass Entire array**

Passing array to function :

- **1. Pass Array element by element**
  - Here individual elements are passed to function as argument.
  - Duplicate **carbon copy of Original variable** is passed to function .
  - So any changes made inside function **does not affects the original value.**
  - Function doesn’t get complete access to the original array element.
  - Function passing method is **“Pass by Value“**

Passing array to function :

- **1. Pass Array element by element**

```
main( )
{
int i ;
int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
for ( i = 0 ; i <= 6 ; i++ )
display ( marks[i] ) ;
}
display ( int m )
{
printf ( "%d ", m ) ;
}
```

Passing array to function :

- **2 . Pass Entire array**
  - Here entire array can be passed as a argument to function .
  - Function gets **complete access** to the original array .
  - While passing entire array Address of first element is passed to function , any changes made inside function , directly **affects the Original value** .
  - Function Passing method : **“Pass by Address“**

Passing array to function :

- **2 . Pass Entire array**

  Now, consider the following function, which takes an array as an argument along with another argument and based on the passed arguments, it returns the average of the numbers passed through the array as follows −

```
double getAverage(int arr[], int size) {

   int i;
   double avg;
   double sum = 0;

   for (i = 0; i < size; ++i) {
      sum += arr[i];
   }

   avg = sum / size;

   return avg;
}
```

Passing array to function :

- **2. Pass Entire array**

  Now, let us call the above function as follows −

```c
#include <stdio.h>

/* function declaration */
double getAverage(int arr[], int size);

int main () {

   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value */
   printf( "Average value is: %f ", avg );

   return 0;
}
```

Const Qualifier:

- The const type qualifier can be applied to an array parameter in a function definition to prevent the original array from being modified in the function body.

```
// Using the const type qualifier with arrays.
#include <stdio.h>

void tryToModifyArray( const int b[] ); // function prototype

// function main begins program execution
int main( void )
{
   int a[] = { 10, 20, 30 }; // initialize array a

   tryToModifyArray( a );

   printf("%d %d %d\n", a[ 0 ], a[ 1 ], a[ 2 ] );
} // end main

// in function tryToModifyArray, array b is const, so it cannot be
// used to modify the original array a in main.
void tryToModifyArray( const int b[] )
{
   b[ 0 ] /= 2; // error
   b[ 1 ] /= 2; // error
   b[ 2 ] /= 2; // error
} // end function tryToModifyArray
```

```
error C2166: l-value specifies const object
error C2166: l-value specifies const object
error C2166: l-value specifies const object
```