CSE101

Arrays

(Arrays and Functions)

Outline

- To declare an array
- To initialize an array
- To pass an array to a function

Introduction

Arrays :

- Collection of *related* data items of *same data type.*
- Static entity – i.e. they remain the same size throughout program execution.

Example where arrays are used:

- to store list of Employee or Student names,
- to store marks of students,
- or to store list of numbers or characters etc.

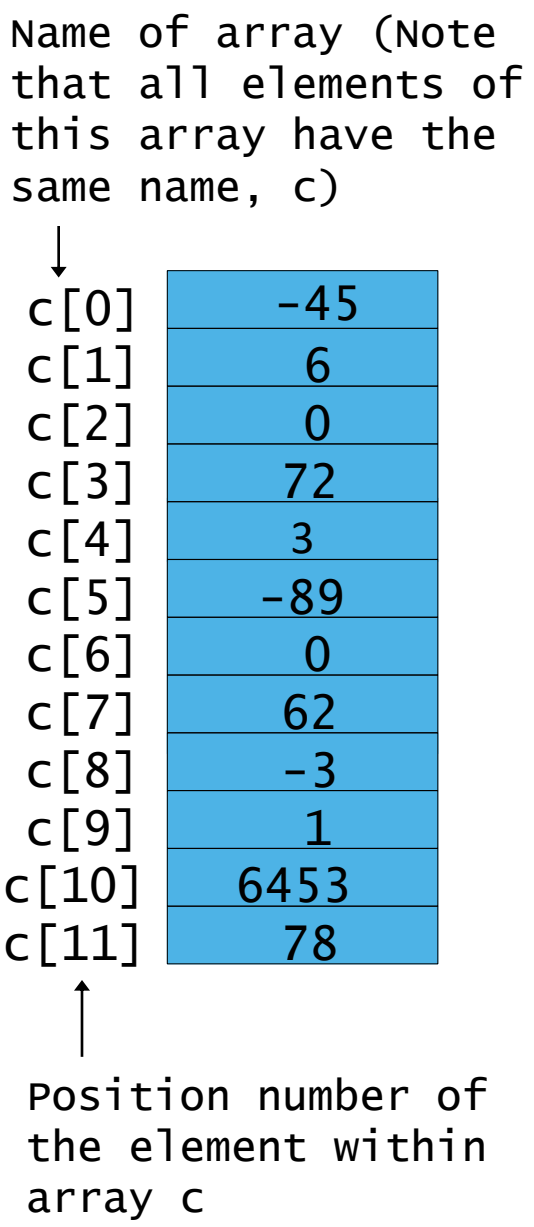
Arrays

Array

- Group of consecutive memory locations
- Same name and data type

- To refer to an element, specify:
  - Array name
  - Position number in square brackets([])
- Format/Syntax:

  arrayname[*position_number*]
  - First element is always at position `0`
  - Eg. `n` element array named `c`:
    - `c[0], c[1]...c[n – 1]`



Important points about Array

- Array data type: any
- Array size: constant value.
- Array Storage: Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Array Examples

- Array elements are like normal variables

```
c[0] =  3;   //stores 3 to c[0] element
scanf ("%d", &c[1]);  //reads c[1] element
printf ("%d, %d", c[0], c[1]); //displays
                                c[0] & c[1] element
```

- The position number inside square brackets is called **subscript/index**.
- Subscript must be integer or an integer expression

```
c[5 + 2] = 7;  (i.e. c[3] = 7)
```

Array Definition

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

    datatype array_Name[number_of_elements];
  - Examples:

    int students[10];

    float myArray[3284];
- Defining multiple arrays of same data type
  - Format is similar to regular variables
  - Example:

    int b[100], x[27];

Array Initialization

- **Initializers**

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

  - If not enough initializers given, then rightmost elements become 0.
  - `int n[5] = { 0 };` `// initialize all elements to 0`
  - C arrays have no bounds checking.
- If size is not mentioned, initializers determine it.

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

  - 5 initializers, therefore 5 element array.

Array Initialization

- Array is same as the variable can take value from the user at run time.
- Array is a group of elements, so **for** loop is used to get the values of every element instead of getting single value at a time.
- Example:

```
int array[5], i;                    // array of size 5
 for(i=0; i<5; i++)
{                                   // loop begins from 0 to 4
    scanf("%d", &array[i]);
}                                   //for loop ends
```

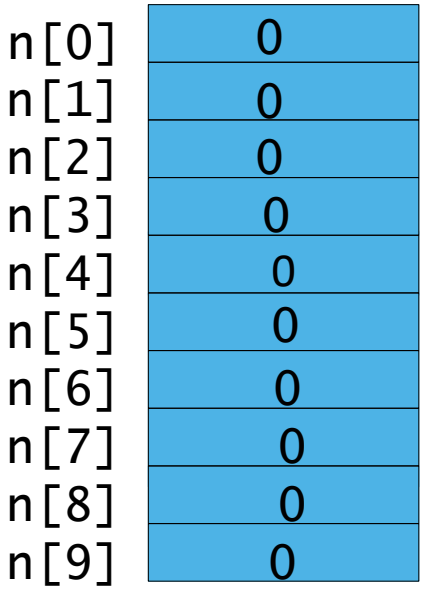
```c
//PROGRAM TO INITIALIZE ARRAY ELEMENTS TO ZERO
#include <stdio.h>
#define MAX_SIZE 10
int main()
{
  int n[ 10 ];       //array declaration
  int i;

  for ( i = 0; i < 10; i++ )
  {
          n[ i ] = 0;       // initialize elements of array n to 0
  }
  printf( "%s, %s \n", "Element", "Value" );

  for ( i = 0; i < 10; i++ )
  {
      printf( "%d, %d \n", i, n[ i ] );
  }
  return 0;
}
```

A macro is a fragment of code that is given a name. You can use that fragment of code in your program by using the name.

These macro definitions allow constant values to be declared for use throughout your code.

OUTPUT:

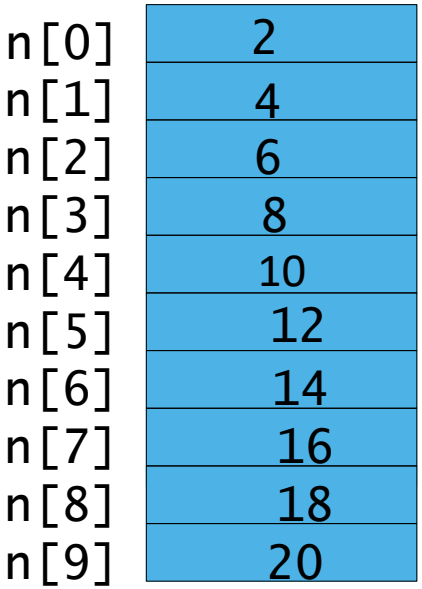
```
Element  Value
   0        0
   1        0
   2        0
   3        0
   4        0
   5        0
   6        0
   7        0
   8        0
   9        0
```



```c
//PROGRAM TO PRINT VALUES OF TABLE OF 2
#include <stdio.h>
#define N 10   // N is size of array
int main()
{
  int table[ N ];       //array declaration
  int i;

  for ( i = 0; i < N; i++ )
  {
            table[ i ] =  2 + 2*i;        // initialize elements of array 2 to 20
  }
  printf(" [%s] \t %s \n", "Index", "Value" );

  for ( i = 0; i < N; i++ )
  {
       printf( "   %d      \t   %d \n", i, table[i] );
  }
  return 0;
}
```

OUTPUT:

```
[Index]         Value
  0               2
  1               4
  2               6
  3               8
  4               10
  5               12
  6               14
  7               16
  8               18
  9               20
```



Q: WAP to take 8 numbers as input from the user and calculate the average using ARRAYS.

```c
//TO PRINT ADDRESS OF THE ARRAY.
#include<stdio.h>
int main()
{
        char abc[10];
        printf("Address of &array[0] abc = %p\n",&abc[0]);
        printf("Address of array abc = %d", &abc);

        return 0;
}
```

OUTPUT:

```
Address of &array[0] abc = 0062FEA6
Address of array abc = 6487718
```

Character Arrays

- Initialized using *string literals*

```
char string1[ ] = "first";
```

It is equivalent to

```
char string1[ ] = { 'f', 'i', 'r', 's', 't', '\0' };
```

  - Null character '\0': terminates strings/ signifies end of string
  - string1 actually has **6 elements.**

– Can access individual characters

string1[3] is character 's'

★ Array name is address of array, so **&** not needed for scanf

```
scanf( "%s", string2 );
```

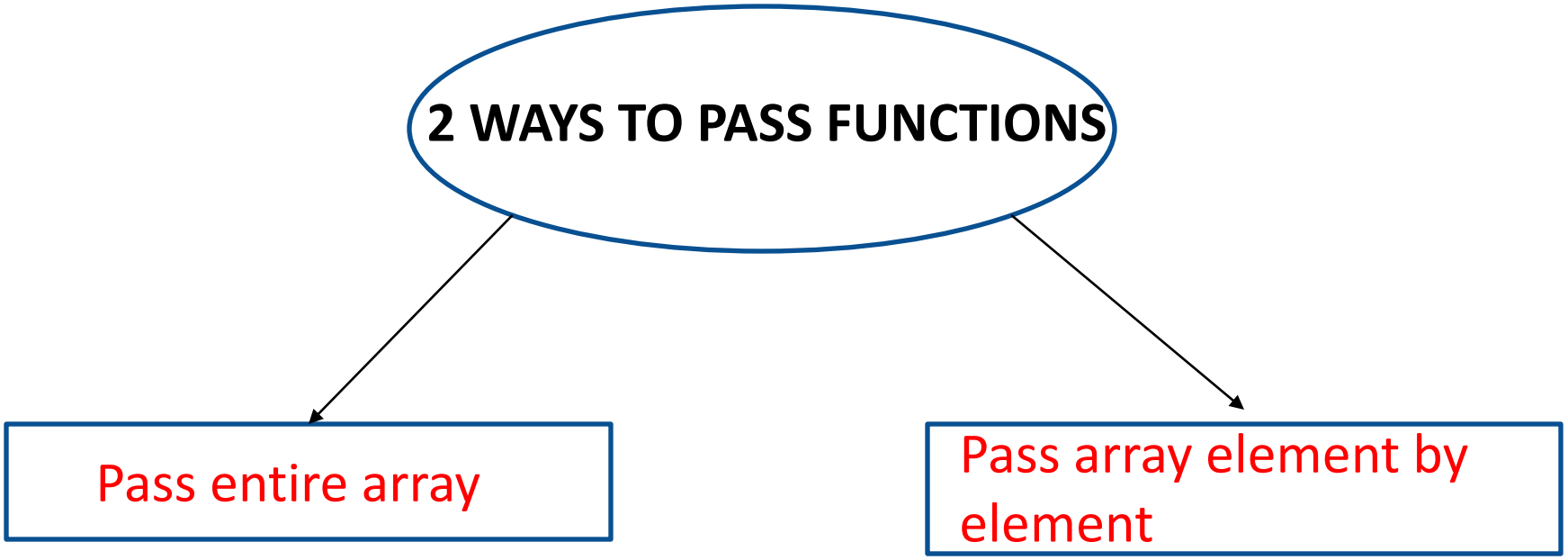
  - Reads characters until whitespace encountered

Program to print character array as strings.

```c
#include <stdio.h>

/* function main begins program execution */
int main()
{
   char string1[ 20 ];                  /* reserves 20 characters */
   char string2[] = "string literal"; /* reserves 15 characters */
   int i;                               /* counter */

   /* read string from user into array string2 */
   printf("Enter a string: ");
   scanf( "%s", string1 );

   /* output strings */
   printf( "string1 is: %s\nstring2 is: %s\n",string1, string2 );
   printf( "\n" );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a string: Hello
string1 is: Hello
string2 is: string literal
```

Passing Arrays to Function



I. Pass entire array

- Entire array can be passed as an argument to the 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.**

```
int modifyArray(int b[], int arraySize);
```

- Function passing method: **"Pass by Address"**

II. Pass array element by element

- Elements are passed to the 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.

```
int modifyElement(int e);
```

- Function passing method: **"Pass by Value"**

Passing Arrays to Functions

- Function prototype

```
void modifyArray(int b[], int arraySize);
```

  - Parameter names optional in prototype
    - int b[] could be written int [ ]
    - int arraySize could be simply int

```
int modifyArray(int [], int);
```

- Function call

```
int a[SIZE];
modifyArray(a, SIZE);
```

Passing complete array to functions

```
#include<stdio.h>
float findAverage(int marks[]);   //function declaration
int main()
{
float avg;
int marks[ ] = {99, 90, 96, 93, 95};
avg = findAverage(marks);    // name of the array is passed as argument.
printf("Average marks = %.1f", avg);
return 0;
}
float findAverage(int marks[])    //function calling
{
int i, sum = 0; float avg;
for (i = 0; i <= 4; i++)
{
sum += marks[i];
}
avg = (sum / 5);
return avg;
}
```

Passing individual array elements to functions

```
#include <stdio.h>
void display(int age)
{
printf("%d", age);
}
int main()
{
int ageArray[] = { 2, 3, 4 };
display(ageArray[2]);   //Passing array element ageArray[2] only.
return 0;
}
```

Next Class:
Applications of Arrays

cse101@lpu.co.in