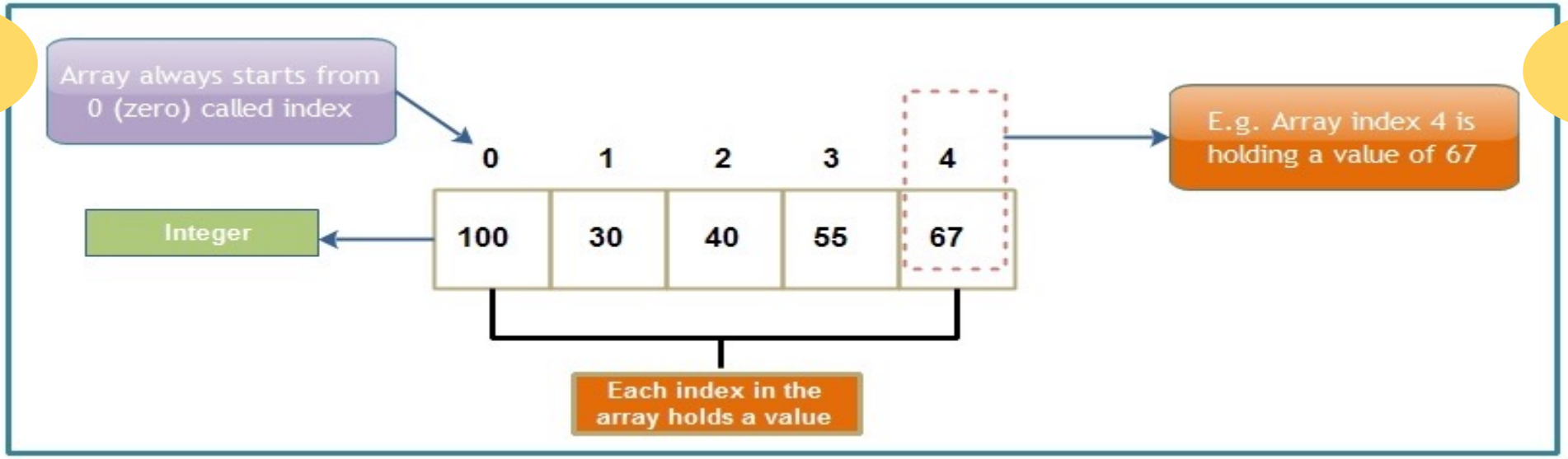
UNIVERSITY of SAN CARLOS
SCIENTIA • VIRTUS • DEVOTIO

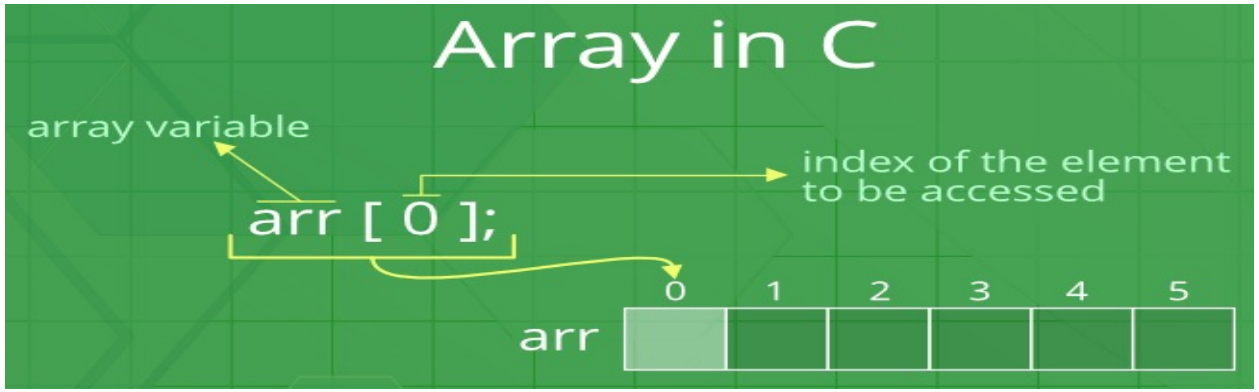
DCISM
DEPARTMENT OF COMPUTER, INFORMATION SCIENCES AND MATHEMATICS

CIS 1101 – PROGRAMMING 1

INTRODUCTION TO ARRAY

ARRAY


Array always starts from 0 (zero) called index
0
1
2
3
4
E.g. Array index 4 is holding a value of 67
Integer
100
30
40
55
67
Each index in the array holds a value


Array in C
array variable
index of the element to be accessed
arr [0];
0
1
2
3
4
5
arr

ARRAY: BASIC CONCEPTS

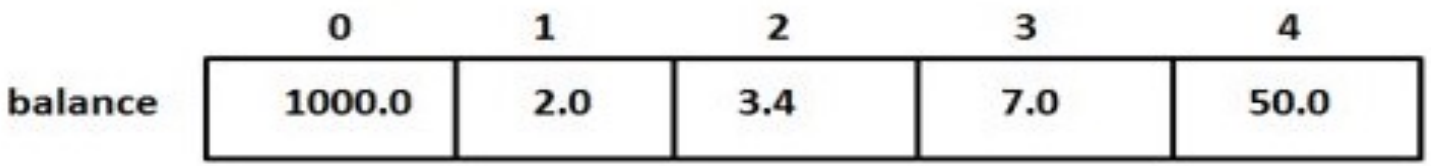
A sequenced collection of elements of the **same data type**.

Called subscripted variables because each element in the array is accessed through the use of **subscript** (or **index**).

A kind of data structure that can store a **fixed-size sequential collection** of elements of the **same type**.

ARRAY: IMPORTANT NOTES

- An array is a derived data type.
- All arrays have **0** as the index of their first element which is also called the **base index** and
- The last index of an array will be total size of the array minus 1.
- The lowest address corresponds to the first element and the highest address to the last element.
- All arrays consist of contiguous memory locations.
- **Example:**



USES OF ARRAYS: SCENARIO 1

❑ Why do we need ARRAY in C?

- ❖ Imagine to have a problem that requires a user to read, process, and print 100 integers.
- ❖ The mentioned numbers must also be kept in memory for the duration of the program.
- ❖ This kind of problem needs a powerful data structure in order to process these large amount of data.

USES OF ARRAYS: SCENARIO 2

❑ Why do we need ARRAY in C?

- ❖ Consider a scenario wherein there is a need to find out the **average of 100 integer numbers** entered by the user.
- ❖ In C, there are **two ways** to do this:

1) Define 100 variables with int data type and then perform 100 scanf() operations to store the entered values in the variables and then at last calculate the average of them.

2) Have a single integer array to store all the values, loop the array to store all the entered values in array and later calculate the average.

USES OF ARRAYS: SCENARIO 2

Which solution is better?

The second solution is better.

It is convenient
to **store same data types**
in one single variable
and later,
access them using array index.

ARRAY: DECLARATION SYNTAX

data _type array _ name [array _ size];

The **array_size**
must be an ***integer constant***
greater than zero
and **data_type**
can be ***any valid C data type***.

The **array_size** is the ***number of elements in the array***.

ARRAY DECLARATION: EXAMPLE

Given:

Consider a **10-element array** called **balance**. Its type is **double**.

Task:

Write the code of the declaration of the array stated above.

Answer:

```
double balance[10];
```

ARRAY DECLARATION: EXAMPLE NOTES

During declaration,
consecutive memory locations
are reserved
for the array and all its elements.

After declaration,
do not assume
that the elements
have initialized to zero.

Random junk value
is at each element's memory location.

ARRAY: BASIC INITIALIZATION

- **Syntax:**

  ```
  data_type array_name[array_size] = {initialization};
  ```

- **Example:**

  ```
  double amount[5] = {975.25,  2.10,  3.45,  7.30,  50.5};
  ```

It is the typical way of initializing an array.

ARRAY: INITIALIZATION WITHOUT SIZE

- **Syntax: data_type array_name[ ] = {initialization};**

- **Example:**

```
int numbers[ ] = {4, 8, 12, 35, 50};
```

When the array is completely initialized, the programmer does not need to specify the size of the array.

ARRAY: PARTIAL INITIALIZATION

- **Syntax:**

  data_type array_name[array_size] = { partial initialization};

- **Example: int distance[5] = {3, 7};**

If the number of values provided
is fewer than the number of elements in the **array**,
the unassigned elements are **filled with zeros**.

ARRAY: INITIALIZATION TO ALL ZEROS

- **Syntax: data_type array_name[array_size] = {0};**
- **Example: int lotsOfNum[1000] = {0};**

It is used to easily initialize an array
to all zeros
and it is done by supplying
just the first zero value (all are filled with zeros).

ARRAY: ACCESSING ELEMENTS

An element is accessed
by indexing the array name.

An array subscript (or index)
is used to access any element
stored in array.

Place the index of the element
within square brackets
after the name of the array.

ARRAY - ACCESSING ELEMENTS: ILLUSTRATION 1

```
double salary = balance[9];
```

The above statement will:

- ✓take the **10th element** from the array **balance**
- ✓and assign the value to **salary** variable.

ARRAY - ACCESSING ELEMENTS: ILLUSTRATION 2

An array mark is declared below:

float mark[5];

mark[0]	mark[1]	mark[2]	mark[3]	mark[4]
79	86	99	60	59

ARRAY - ACCESSING ELEMENTS: ILLUSTRATION 2

- The first element is **mark[0]**, second element is **mark[1]** and so on.
- Arrays have **0** as the first index, **not 1**.
- If the size of an array is **n**, to access the last element, (**n-1**) index is used.
- Suppose the starting address of **mark[0]** is **2120d**, then the next address **mark[1]** will be **2124d**, address of **mark[2]** will be **2128d** and so on (because the **size of a float is 4 bytes**).

ARRAY – ACCESSING ELEMENTS: PROBLEM

Given:

A program will ask the user to enter six integers and store them in an array. The program will print the elements stored in the array. Then it will change the element of the index 3 of the array into 98 and will print the new set of elements.

Task:

Write the code of the program described above.

ACCESSING ARRAYS AS FUNCTION ARGUMENTS

- **Passing Arrays as Function Arguments in C:**
  - There are multiple ways to pass one-dimensional arrays as arguments in C.
  - We need to pass the array to a function to make it accessible within the function.
  - If we pass an entire array to a function, all the elements of the array can be accessed within the function.
  - Single array elements can also be passed as arguments.
  - This can be done in exactly the same way as we pass variables to a function.

EXAMPLE 1: PASSING ARRAY DIRECTLY TO FUNCTION

```
#include<stdio.h>

void printArray(int arr[ ], int size);
int main()
{
      int arr[5] = {1,2,3,4,5};
      printArray(arr, 5);  /* Pass array directly to function printArray */
      return 0;
}
void printArray(int arr[ ], int size)
{
      int i;
      printf("The array elements are: ");
      for(i = 0; i < size; i++) {  printf("%d  ", arr[i]);  }
}
```

ACCESSING ARRAYS AS FUNCTION ARGUMENTS

Important Note:

- Arrays in C are passed as reference not by value.
- This means that any changes to array within the function will also persist outside the function.

EXAMPLE 2: PASSING AN ARRAY TO A FUNCTION

```
#include <stdio.h>          /* Program to calculate average  */
float average(float age[]);
int main()
{
      float avg, age[ ] = {23.4, 55.75, 22.6, 3.2, 40.5, 18.9};
      avg = average(age); /* Only name of an array is passed as an argument */
      printf("The average age = %.2f", avg);
      return 0;
}
float average(float age[ ])
{
      int i;
      float avg, sum = 0.0;
      for (i = 0; i < 6; ++i) {    sum += age[i];    }
      avg = (sum / 6);
      return avg;
}
```

EXAMPLE 3: PASSING AN ARRAY TO A FUNCTION

```
#include<stdio.h>
double getAverage(double arr[], int size);
int main() {
      double avg, age[6] = {23.4, 55.75, 22.6, 3.2, 40.5, 18.9};
      avg = getAverage(age,6);
      printf("The average age = %.2lf", avg);
      return 0;
}
double getAverage(double arr[ ], int size) {
      int i;
      double avg, sum = 0;
      for(i=0; i<size; i++){ sum +=arr[i];  }
      avg = sum / size;
      return avg;
}
```

ONE-DIMENSIONAL ARRAY

- It is **a list of related values** in which all items in it **have the same data type**.
- All list members are **stored using single group name**.
- It is **a type of linear array** and accessing its elements involves a **single subscript** which can **either represent a row or column index**.

ONE-DIMENSIONAL ARRAY

- **Example:**

  A list of grades: 97, 89, 76, 91, 82

- **Declaration & Initialization:**

  int grades[5] = {97, 89, 76, 91, 82};

RETURNING ARRAY FROM FUNCTIONS

C programming does not allow to return an entire array as an argument to a function. However, you can return a pointer to an array by specifying the array's name without an index.

EXAMPLE: RETURNING ARRAY FROM FUNCTION

```c
#include <stdio.h>
int* createArray();

int main() {
    int *n;
    n=createArray();
    for(int i=0;i<5;i++) {
        printf("%d ", n[i]);
    }
    return 0;
}
```

```c
int* createArray(){
    int arr[5];
    arr[0] = 1;
    arr[1] = 2;
    arr[2] = 3;
    arr[3] = 4;
    arr[4] = 5;
    return arr;
}
```

PASSING ARRAY TO FUNCTION (PASS BY ADDRESS)

```
#include <stdio.h>
void arrayChange(int*);

int main() {
    int arr[5] = {1,2,3,4,5};

    for(int x = 0; x < 5; x++){
        printf("%d ",arr[x]);
    }
    arrayChange(arr);
    printf("\n");
    for(int x = 0; x < 5; x++){
        printf("%d ",arr[x]);
    }

    return 0;
}
```

```
void arrayChange(int*arr){
    arr[0] = 5;
    arr[1] = 4;
    arr[2] = 3;
    arr[3] = 2;
    arr[4] = 1;
}
```

Output:

1 2 3 4 5

5 4 3 2 1