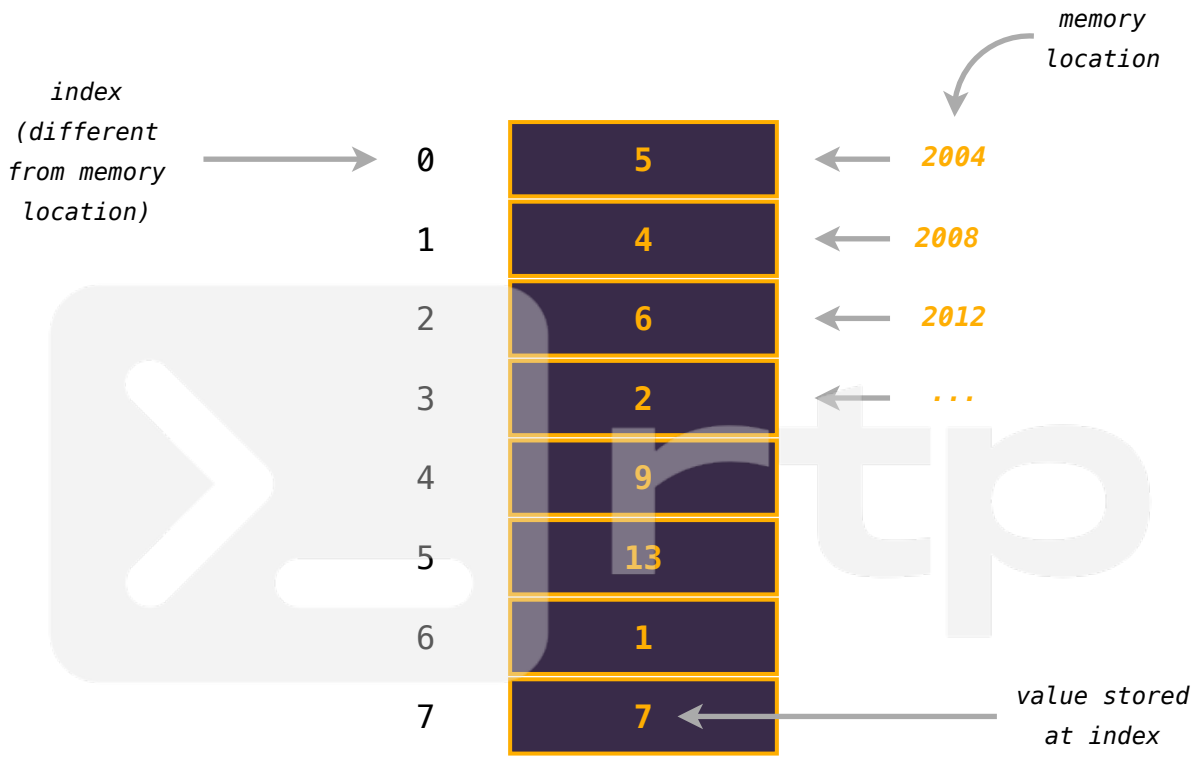
week 3_2:

arrays

rajin teaches programming

VISUALIZATION OF AN ARRAY



An array is a **sequential container** that **stores any** kind of data. It is **not** a **data structure** in **itself.** So, in theory, we could **declare** an **array of** integers, floats, characters, or even strings. Each **cell** will then therefore take up a **space** of whatever the **data type** is. In this case, it is taking up **4 bytes** for each cell of the array (as each **int data type** is **4 bytes**). Arrays are **conventionally** numbered **from 0,** so we can conclude that the **last** value is stored at **size-1.**

INTRODUCTION TO ARRAYS:

Arrays are a **fundamental concept** in programming that allow us to store **multiple values** of the **same** data type **under** a **single name.** They provide a convenient way to work with **collections** of **data** and are widely used in various applications.

– DECLARATION AND INITIALIZATION:

In C programming, arrays are **declared** using the following syntax:

ARRAY DECLARATION

```
datatype arrayName[arraySize]
```

Here, `**datatype**` specifies the **type** of elements in the array, `**arrayName**` is the **name** of the array, and `**arraySize**` is the **number of elements** in the array.

In C, as you already know, **variables** that are **declared** are given **random** values. The **same** works with **arrays** whenever we **allocate space** for one in the memory. Thus, **after declaring** an array, we **need** to **initialize it** too, either **directly** or by using a **loop.**

ARRAY INITIALIZATION

```
datatype arrayName[arraySize] = {value1, value2, ..., valueN};
```

– ACCESSING ARRAY ELEMENTS:

Individual elements of an array can be **accessed** using the **array index.** The index **starts from 0** for the **first element** and goes **up to** `**arraySize – 1**` for the **last element.**

ACCESSING ARRAY ELEMENTS

```
arrayName[index]
```

Developer - arrays.c

```c
#include<stdio.h>

int main() {

    int list[6] = {2, 4, 5, 6, 4, 9};

    int firstElement = list[0];
    int lastElement = list[5];

    printf("The first element is: %d, and the last element is: %d\n", firstElement, lastElement);

    return 0;
}
```

an example of array declaration, initialization, and accessing.

ARRAYS AND LOOPS:

Arrays are often used in **conjunction** with **loops** to perform operations on **multiple elements** efficiently. We can use loops to **access** and **modify** array elements. Since we have already covered for loops, this should be easy to understand. Examples below show **both accessing** array elements and **inserting into** arrays.

– INSERTING ARRAY ELEMENTS USING LOOPS:

a snippet of code,

Developer - array_output.c

```c
#include<stdio.h>

int main() {

    int size = 10;
    int array[size];

    for (int i=0; i<size; i++) {

        scanf("%d", &array[i]);
    }
```

showing how to insert array elements using a for loop.

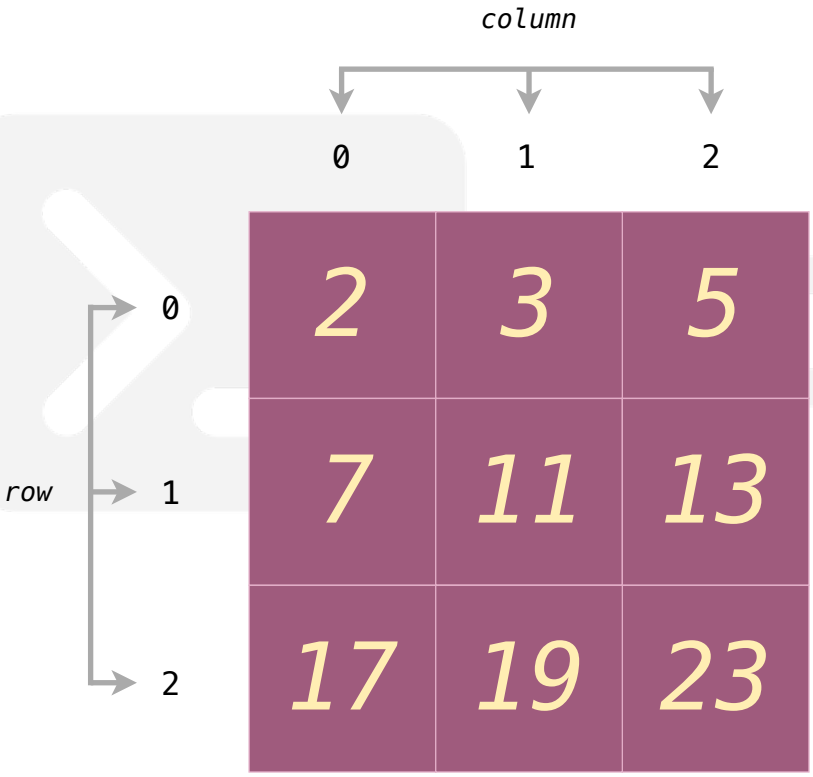
– ACCESSING ARRAY ELEMENTS USING LOOPS:

other half of the same snippet,

```
        }

        for (int i=0; i<size; i++) {

            printf("Element at index[%d]: %d\n", i, array[i]);
        }

        return 0;
}
```

showing how to access array elements using a for loop.

MULTI-DIMENSIONAL ARRAYS:



All the arrays discussed so far have been a **one-dimensional** array, where they store data **sequentially**, in a **single column**, perhaps. However, **two dimensional arrays**, and even **3 dimensional arrays** exist. For ease of understanding, two dimensional arrays are often referred to as a **matrix.** The **visualization** is very accurate and helps us to understand where **each items** in a 2-D array go. Two dimensional arrays are often **divided** into **rows** and **columns**, **numbered**, again from **0 to n.**

– DECLARATION OF A 2–D ARRAY (MATRIX):

DECLARING 2–D ARRAY

```
arrayName[rowSize][colSize]
```

– INSERTING AND ACCESSING ELEMENTS IN 2–D ARRAY:

Like shown previously, **elements** can either be **inserted directly**, or by using a **for loop** in a **two–dimensional** array. The **same** concept can be used for **printing** it too. Inserting **directly** has some other **issues** too, as will be explained in class, so let us look at **insertion** and **printing** for a **2–D array:**

initialization

nested loop for insertion

nested loop for printing

Developer - matrix.c

```c
#include<stdio.h>

int main() {

    int rows = 3;
    int cols = 3;
    int array[rows][cols];

    for (int i=0; i<rows; i++) {

        for (int j=0; j<cols; j++) {

            printf("Enter item at [%d][%d]: ", i, j);
            scanf("%d", &array[i][j]);
        }
    }

    printf("\nThe items in your matrix are: \n");

    for (int i=0; i<rows; i++) {

        for (int j=0; j<cols; j++) {

            printf("%d\t", array[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

CONCLUSION:

Arrays are **powerful containers** in C programming that enable **efficient storage** and **manipulation** of **collections** of **data.** Understanding arrays is essential for mastering various programming tasks**.** C provides **several functions** for **manipulating arrays,** such as **sorting, searching,** and **copying.** These functions are available in the **standard library `<stdlib.h>`** and **`<string.h>`.**

PRACTICE QUESTIONS:

- Write a C program to find the sum of elements in a one-dimensional array**.**
- Write a C program to find the largest element in a one-dimensional array**.**
- Write a C program to sort elements of a one-dimensional array in ascending order**.**
- Write a C program to perform matrix addition of two 2D matrix arrays**.**

next class 3_3:

strings

rajin teaches programming

reach out for classes at: rajin.khan2001@gmail.com