Week 4
Arrays
(Summary on Key Points)

1

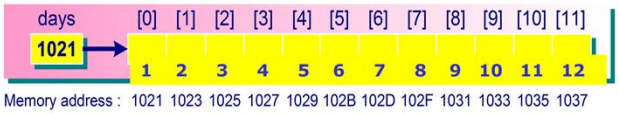
Arrays

- One-Dimensional Arrays
- Pointer Constants vs Pointer Variables in Arrays
- Processing 1D Arrays within a Function
  - Using Indexes and Using Pointers
- Processing 1D Arrays in Function Communications
  - Using Indexes and Using Pointers
- Two-Dimensional Arrays
- Processing 2D Arrays within a Function
  - Using Indexes and Using Pointers
- Processing 2D Arrays in Function Communications
  - Using Indexes and Using Pointers

2

1D Arrays

int days[12]; /*array of 12 integers*/



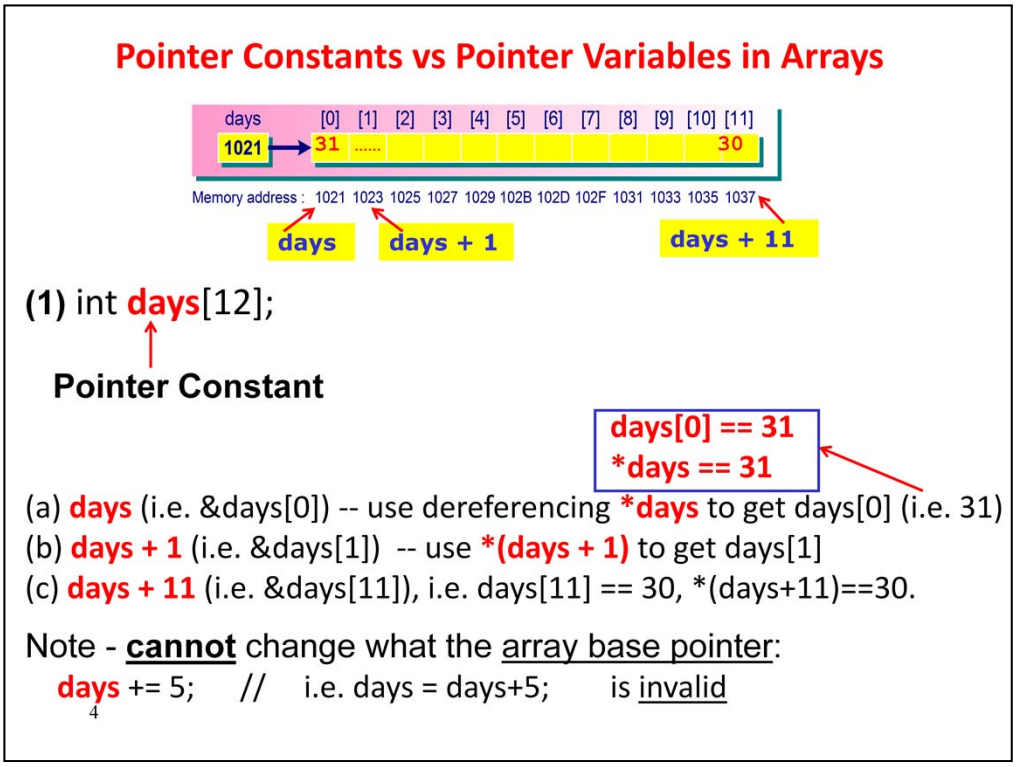
What are they?

- **days?** => contains the address (or pointer) of the **1st element of the array** (i.e. 1021)
- **&days?** => memory address of the array variable (e.g. 1234)
- ***days?** => dereferencing the array to get the value (i.e. days[0], the value of 1); ***(days+1) == days[1], etc.**
- **days[0], days[1], days[2], etc.** => the values stored at the respective array index locations, the values of 1, 2, 3, etc.
- **&days[0], &days[1], &days[2], etc.** => the memory addresses of the memory locations of the respective array index locations, the memory addresses of 1021, 1023, etc.

3

1D Arrays

1. The array name by itself, **days**, is the address (or pointer) of the 1st element of the array.
2. What have you observed?
   - The array variable **days** contains a pointer constant (i.e. 1021) (i.e. the value cannot be changed)
   - The array index **days[0]**, days[1], etc. contains the array value at that index location.
   - The array element address is **&days[0] (i.e. 1021)**, &days[1], etc. That is, days[0] has the address of 1021, days[1] has the address of 1023, etc.
3. The goal is to use the pointer constant **days** for accessing each array element.



Pointer Constants

1. The array name **days** is a pointer constant.
2. Since the array name is the pointer to the first element of the array, we have

   days == &days[0]

   days+1 == &days[1]

   days+i == &days[i]
3. Therefore, there are two ways to retrieve the content of the element of the arrays. For example, if we want to get the value of the first element, we can use either

   days[0] // using index notation or

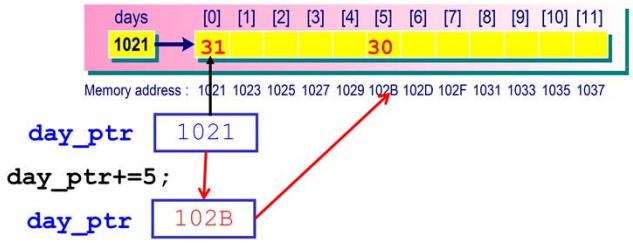
   ***days // using pointer notation**
4. For example, we can write ***(days+1)** to access the array element **days[1]**. Similarly, ***(days+2)** is used to access array element **days[2]**, etc.
5. However, it is important to note that the array name is a **pointer constant**, not a pointer variable. It means that the value stored in **days** cannot be changed by any statements. As such, the following assignment statements are invalid:

   days += 5; and

   days++;

Pointer Constants vs Pointer Variables in Arrays

int **days**[MTHS]= {31,28,31,30,31,30,31,31,30,31,30,31};



(2) int ***day_ptr**;

Pointer Variable [can store different addresses]

(a) day_ptr = days; // valid

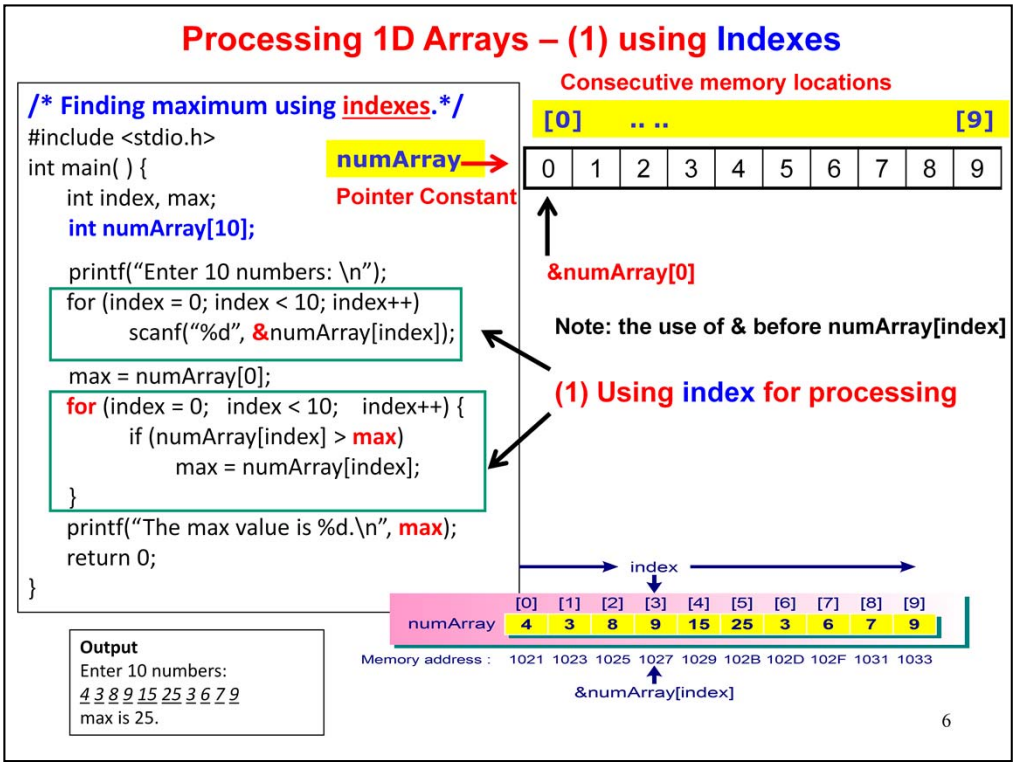
(b) **day_ptr += 5;** // valid – update the pointer variable

➔ dereferencing: ***day_ptr** will get **30**

5

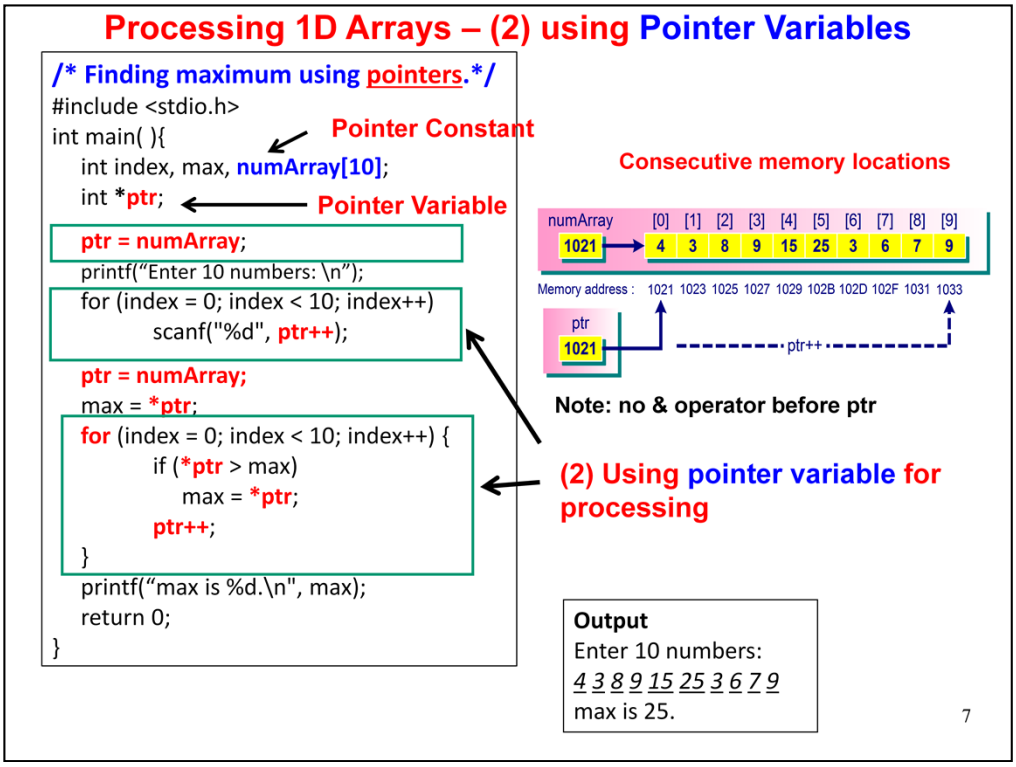
Pointer Variables

1. A pointer variable can take on different addresses.
2. In the program, we declare an array variable **days**[] of 12 elements and initialized it with values: int **days**[MTHS]= {31,28,31,30,31,30,31,31,30,31,30,31}; where **days** is a pointer constant which is declared as an array of 12 elements.
3. Then, we declare an integer pointer variable **day_ptr**: int ***day_ptr**;
4. The statement **day_ptr = days;** assigns the value 1021 from the array variable **days** to the pointer variable **day_ptr**. This causes the pointer variable to point to the first element of the array.
5. After that, we can use the pointer variable **day_ptr** to access each element of the array.



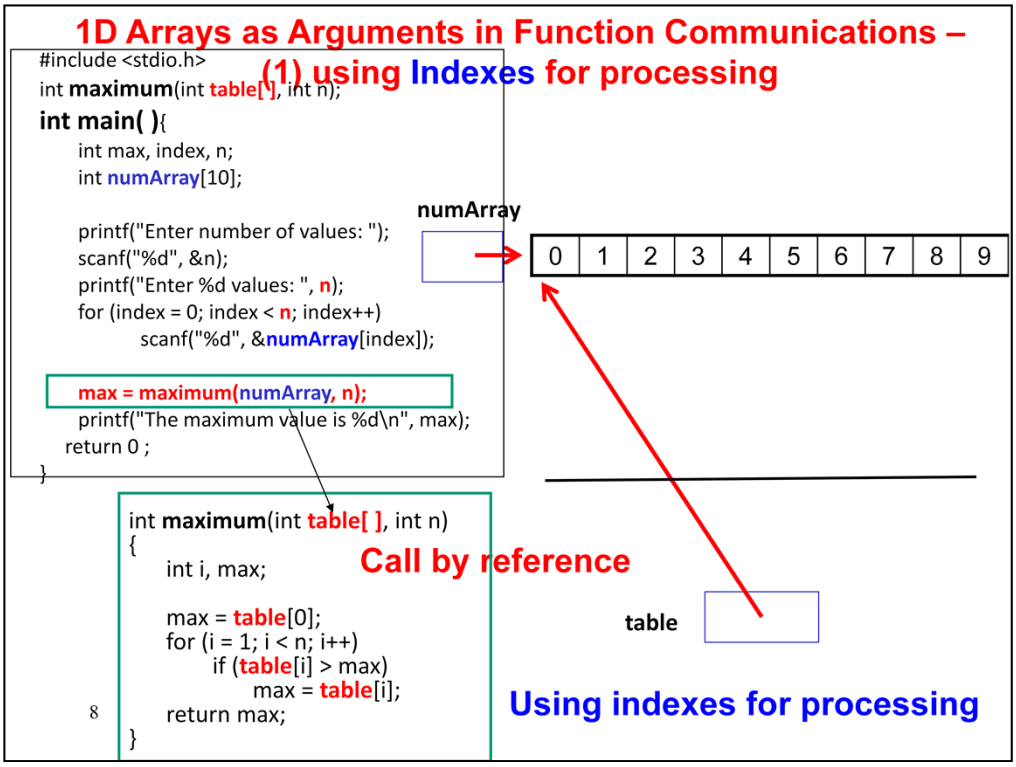
Traversing an Array – Finding the Maximum Value

1. The program finds the maximum non-negative value in an array. The value for each item in an array is read from the user and stored in the array. Then, the array is traversed element by element in order to find the maximum value in the array.
2. In the program, the value for each item in an array is firstly read from the user and stored in the array. The value –**1** is assigned to the variable **max,** which is defined as the current maximum.
3. Then, the items in the array are checked one by one using a **for** loop. If the value of the next item is larger than the current maximum, it becomes the current maximum. If the value of the next item is less than the current maximum, the current value of **max** is retained. The maximum value in the array is then printed on the screen.



Find Maximum: Using Pointer Variables

1. The previous program uses the pointer constant **numArray** to access all the elements of the array.
2. Another way to access the elements of an array is to use a pointer variable. This program gives an example using a pointer variable to find the maximum element of the array.
3. To achieve this, it is important to assign **numArray** to **ptr**: **ptr = numArray;** After that, we can read in the array data via the pointer variable **ptr**.
4. In the first **for** loop, we use **scanf()** to read in user input. We increment the **ptr** as **ptr++;** to access each element of the array in order to store the input integer into the corresponding index location of the array. The first input will be stored at index location **numArray[0]**, after increasing the pointer **ptr** by 1, the next input integer will be stored at location **numArray[1]**, etc.
5. To find the maximum value stored in the array, we also use a **for** loop. In the second **for** loop, it traverses each element in the array using the pointer variable **ptr**. The value stored at the location of the array is referred to as ***ptr**. The content of each element of the array is compared with the current maximum value. After executing the loop, the maximum value in the array is determined. And the variable **max** will store the maximum value.



Implementing Maximum: Using Array Indexes

1. The implementation of the function **maximum()** uses **array indexes**. It has two parameters: **table** and **n**.
2. The array is traversed element by element using indexing with **table[i]**, where **i** is the index from 0 to **n**-1, in order to find the maximum number.
3. At the end of the function, the maximum number stored in **max** is passed back to the calling function.

1D Arrays as Arguments in Function Communications – (2) using pointer variables for processing

```
int maximum(int table[ ], int n)
{
    int i, max;

    max = *table;
    for (i = 1; i < n; i++)
            if (*(table+i) > max)
                    max = *(table+i);
    return max;
}
```

Using pointer

```
int maximum(int *table, int n)
{
    int i, max;

    max = table[0];
    for (i = 1; i < n; i++)
          if (table[i] > max)
                max = table[i];
    return max;
}
```

Using index

```
int maximum(int table[ ], int n)
{
  int i, max;
  max = *table;
  for (i = 0; i < n; i++) {
    if (*table > max)
      max = *table;
    ++table;
  }
  return max;
}
```

Using pointer

```
int maximum(int *table, int n)
{
  int i, max;
  max = *table;
  for (i = 0; i < n; i++) {
    if (*table > max)
      max = *table;
    ++table;
  }
  return max;
}
```

Using pointer

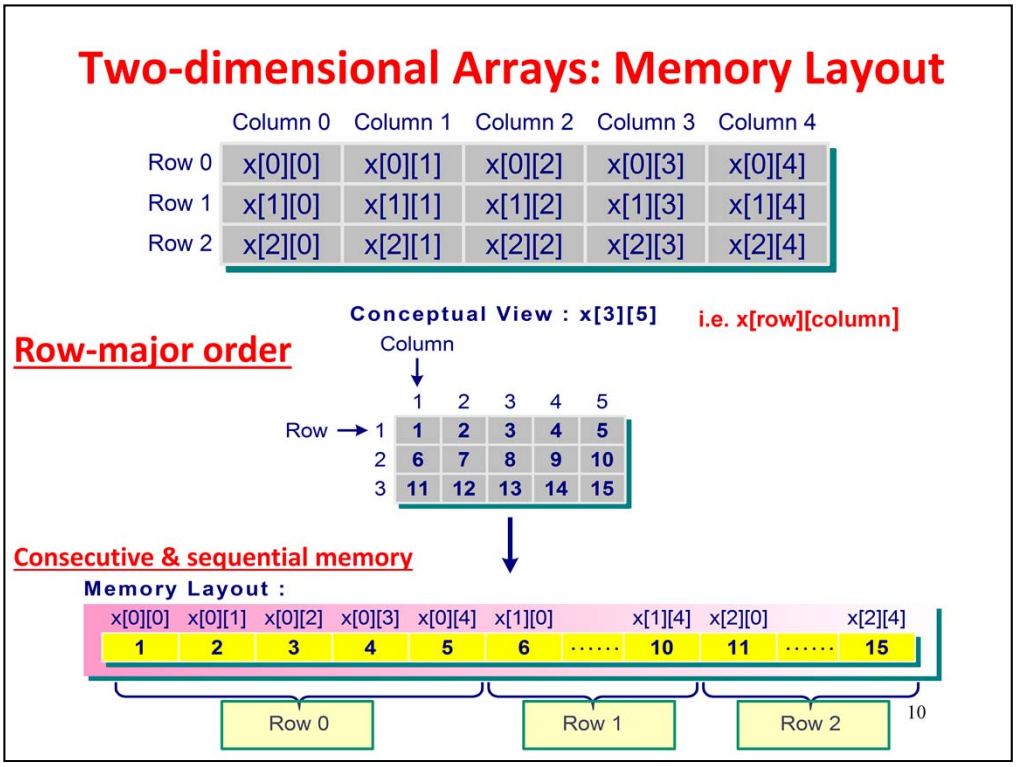
9

Implementing Maximum: Using Array Base Address

1. The implementation of the function **maximum()** uses array base address.
2. As shown, the base address of the array **table** is used. When traversing the array, the array element is accessed via ***(table+i)**, where **i** is the index from 0 to **n**-1.
3. The maximum number is then determined at the end of the loop.

Implementing Maximum: Using Pointer Variable

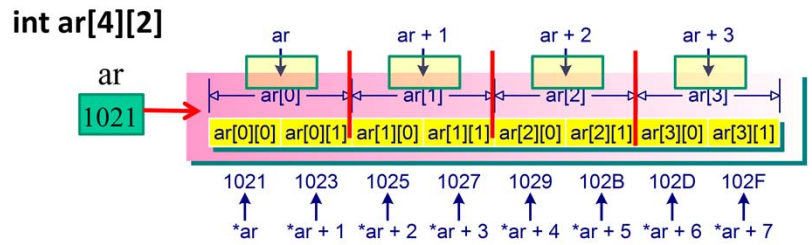
1. The implementation of the function **maximum()** uses pointer variable notation.
2. In this version of implementation, the array **table** is used as a pointer variable. When traversing the array, the array element is accessed via ***table**, and the variable **table** is incremented by 1 using **table++** in order to access each element of the array.
3. The maximum number is then determined at the end of the loop.



Two-dimensional Arrays: Memory Layout

1. The statement **int x[3][5];** declares a two-dimensional array **x[][]** of type **int** having three rows and five columns. The compiler will set aside the memory for storing the elements of the array.
2. The two-dimensional array can also be viewed as a table made up of rows and columns. For example, the array **x[3][5]** can be represented as a table. The array consists of three rows and five columns.
3. The array name and two indexes are used to represent each individual element of the array. The first index is used for the row, and the second index is used for column ordering. For example, **x[0][0]** represents the first row and first column, and **x[1][0]** represents the second row and first column, and **x[1][3]** represents second row and fourth column, etc.
4. A two-dimensional array is stored in ***row-major*** order in the memory.
5. Note that the memory storage of the two-dimensional array **x[3][5]** is consecutive and sequential.

Two-dimensional Arrays and Pointers



Two ways to access two-dimensional Array:

- Using indexes (easy and simple): e.g. **ar[m][n]**
- Using pointers and the general formula for 2D array:
  ar[m][n] == *(*(ar + m) + n)

11

Two-dimensional Arrays and Pointers

1. There are two ways to access each element of the array :
   a) Using the index approach: **ar[m][n]** with indexes **m** and **n**; or
   b) Using the general formula: ***(*(ar+m)+n)** for **ar[m][n]**.

Processing Two-dimensional Arrays: Using Indexes and Pointers

```
#include <stdio.h>
int main() {
  int ar[3][3]= {
    {5, 10, 15},
    {10, 20, 30},
    {20, 40, 60}
  };
  int index, i, j;
    // (1) using indexes
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      printf("%d ", ar[i][j]);
  printf("\n");
    // (2) using the general formula
  for (i=0; i<3; i++)
    for (j=0; j<3; j++)
      printf("%d ", *(*(ar+i)+j));
  return 0;
}
```

Output

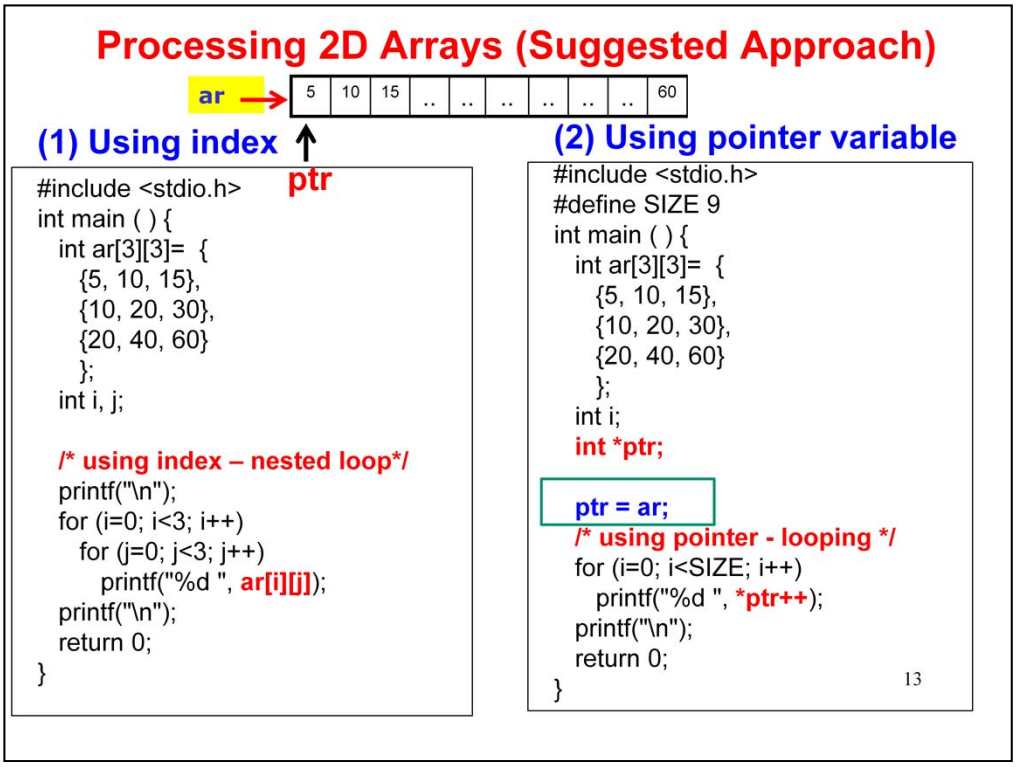
```
5 10 15 10 20 30 20 40 60
5 10 15 10 20 30 20 40 60
```

Note that using indexes will be easier

12

Processing Two-dimensional Arrays: Example

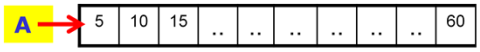
1. The program aims to print the value of each array element in a two-dimensional array.
2. In the program, it first initializes each array element of the array **ar[3][3]**.
3. There are two ways to access each element of the array with a nested **for** loop:
   - Using the index approach or
   - Using the general formula



Processing Two-dimensional Arrays: Suggested Approaches

1. For processing two-dimensional arrays, you may use array index or pointer variable for processing each element of the array.
2. When using the index approach, indexes are used to access each individual element of a two-dimensional array.
3. When using pointer variable approach, a pointer variable is declared and assigned with the array value. It is then used to traverse each element of a two-dimensional array by incrementing the pointer variable to access the content of each element of the array.

2D Arrays as Arguments in Function Communications




```
#include <stdio.h>
void minMax(int a[5][5], int *min, int *max);

int main()
{
   int A[5][5];
   int i, j;
   int min, max;

   printf("Enter your matrix data (5x5): \n");
   for (i=0; i<5; i++)
      for (j=0; j<5; j++)
         scanf("%d", &A[i][j]);
   minMax(A, &min, &max);
   printf("min = %d; max = %d", min, max);
   return 0;
}
```

(1) Using index

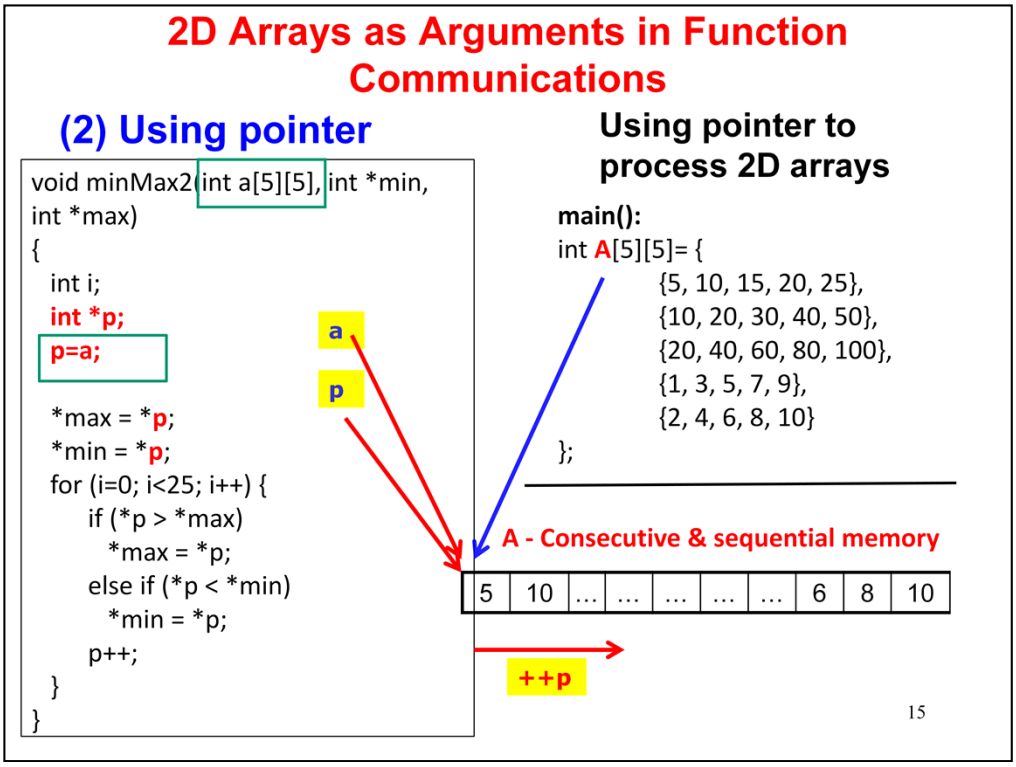
```
void minMax(int a[5][5],
            int *min, int *max)
{
    int i, j;

    *max = a[0][0];
    *min = a[0][0];
    for (i=0; i<5; i++)
       for (j=0; j<5; j++) {
          if (a[i][j] > *max)
             *max = a[i][j];
          else if (a[i][j] < *min)
             *min = a[i][j];
       }
}
```

14

minMax: Using the Array Index Approach

1. In this implementation using the array index approach, a nested **for** loop is used to process the two-dimensional array in the function.
2. It first initializes the ***max** and ***min** to store the first array element number.
3. The two-dimensional array **a** is processed using indexes to access and compare all the elements stored in the array with ***max** and ***min**.
4. After the processing of the two-dimensional array, the maximum and minimum numbers are determined and stored at ***max** and ***min** respectively.
5. The values of minimum and maximum are returned to the calling function via call by reference.
6. The implementation using indexes is quite straightforward.



minMax: Using the Pointer Variable Approach

1. Different from the previous approach using the base address, we can also use the pointer variable approach by updating the **pointer variable** directly.
2. Similarly, a **for** loop is used to traverse and process the two-dimensional array by treating it as an one-dimensional array.
3. The index variable is not needed in this approach. We can update the pointer variable to the corresponding array memory location by updating the pointer variable p, i.e. **p++**, and retrieves the array element content via ***p**. Each array element content will be compared with ***max** and ***min** to determine the maximum and minimum numbers respectively. At the end of the processing, the maximum and minimum numbers are determined and stored at ***max** and ***min** respectively.
4. The minimum and maximum values are returned to the calling function via call by reference.