

SC1003 Review Lecture Week 10





Review Lecture – Week 10

- **Week 10 – Learning Materials**
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
- Reviews on Arrays
- Examples
- Assignment Submission and Grading



Learning Schedule (Week 8 – Week 13)

Week	Week 8 4 Oct	Week 9 11 Oct	Week 10 18 Oct	Week 11 25 Oct	Week 12 1 Nov	Week 13 8 Nov	Week 14 15 Nov
Topics	Basic C Programming and Control Flow	Functions and Pointers	Arrays	Character Strings	Structures		
Review Lecture	Date: 4 Oct 2021 (Monday) Time: 9:30am-10:30am Online: MS Teams (the link for the online lecture is given at the end of the table)	Date: 11 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 18 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 25 Oct 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)	Date: 1 Nov 2021 (Mon) Time: 9:30am-10:30am Online: MS Teams (see below for the link for online lecture)		Lab Test (MCQ Test & Coding Test) Dates: 15 Nov (Mon) and 16 Nov (Tue) Details will be announced when confirmed.
e-Learning Lectures	Learn: Course Introduction Learn: (1) Basic C Programming; (2) Control Flow	Learn: (1) Functions and (2) Pointers	Learn: (1) 1-D Arrays and (2) 2-D Arrays	Learn: Character Strings	Learn: Structures		
Lab-Tutorial	Learn: CodeBlocks IDE Do: Lab-Tutorial 1 (Qns are also available in APAS)	Do: Lab-Tutorial 2 (Qns are also available in APAS)	Do: Lab-Tutorial 3 (Qns are also available in APAS)	Do: Lab-Tutorial 4 (Qns are also available in APAS)	Do: Lab-Tutorial 5 (Qns are also available in APAS)		
Practice Questions	Learn: using APAS system Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)	Do: Coding Practice Questions (APAS>Quiz) Do: MCQ Questions (LAMS)		
Assignment	Learn: (1) Assignment Submission and Grading process; (2) Review Request Form (Procedure)		Assignment paper – Available in APAS			Assignment due	



Lecture Video - Arrays

- Watch Lecture Video – 1-D Arrays and 2-D Arrays
(NTULearn: C Programming > E-Learning Lectures > Week 10)



5.1 - 1-D Arrays - Lecture

5.1 - 1-D Arrays - Lecture (33:43)

5.1
One-dimensional
Arrays



5.2 - 2-D Arrays - Lecture

5.2 - 2-D Arrays - Lecture (30:49)

5.2
Two-dimensional
Arrays



Week 10 - Lab

- Lab 3 – Arrays

(NTULearn: C Programming > Lab-Tutorials > Lab-Tutorial 3)

Lab 3 - Arrays

Lab session – One hour is scheduled for the lab session. There are 2 questions for this lab. In addition, there is also 1 practice question for you to try if you have extra time in the lab.

Note: You do not need to submit your code for this lab.

Lab Questions

1. (**findAr1D**) Write a function **findAr1D()** that returns the subscript of the **first appearance** of a target number in an array. For example, if `ar = { 3,6,9,4,7,8 }`, then **findAr1D(6,ar,3)** will return 0 where 6 is the size of the array and 3 is the number to be found, and **findAr1D(6,ar,9)** will return 2. If the required number is not in the array, the function will return -1. The function prototype is given as follows:

Lab Coding Questions: APAS > Exercise

1. findAr1D
2. reverseAr1D
3. swap2RowsCols

Suggested solutions:

Available in the same folder.
You may refer to the suggested code if you have any difficulty in attempting the lab questions.

Available at: APAS > Exercise (choose Topic): You may test your code with sample test cases in APAS.



Week 10 - Tutorial

- Tutorial 3 – Arrays**

(NTULearn: C Programming > Lab-Tutorials > Lab-Tutorial 3)

Tutorial 3 – Arrays

1. Explain how the addition of 1 to every element of the two dimensional array 'array' is done in the following program. What if the for statement at 'line a' is replaced by this statement:

```
add1(array[0], 3 * 4);
```

```
#include <stdio.h>
void add1(int ar[], int size);
int main()
{
    int array[3][4];
    int h,k;
```

Tutorial Coding Questions:

1. transpose2D
2. reduceMatrix2D

Suggested solutions:

Available at the end of each week in the same folder in NTULearn.

Available at: APAS > Exercise (choose Topic): You may test your code with sample test cases in APAS.



LAMS MCQ Questions

- LAMS MCQ Questions – 1-D Arrays and 2-D Arrays**
(NTULearn: C Programming > LAMS MCQ Questions > 1-D Arrays)

LAMS LAMS MCQ Practice Questions - One-dimensional Arrays

Watch the lecture video and read the lecture notes before attempting the MCQ practice questions are for exercise only.

One-dimensional Arrays

Q1

What will be the output of the program?

```
#include <stdio.h>
int main()
{
    int i,a[10];
    for (i=9;i>=0;i--) a[i]=10-i;
    printf("%d %d %d",a[3],a[6],a[9]);
    return 0;
}
```

- A. 2 5 8
- B. 7 4 1
- C. 8 5 2
- D. 3 6 9

LAMS LAMS MCQ Practice Questions - Two-dimensional Arrays

Watch the lecture video and read the lecture notes before attempting the MCQ practice questions are for exercise only.

Two-dimensional Arrays

Q1

What will be the output of the program?

```
#include <stdio.h>
int main()
{
    int i,j,a[2][3]={ {2,4,6}, {8,10,12} };
    for (i=0;i<3;i++)
    {
        for (j=0;j<2;j++)
            printf("%d ", a[j][i]);
        printf("\n");
    }
    return 0;
}
```

- Answers and explanations on each question are available in the same folder.



APAS - Coding Practice Questions

- Coding Practice Questions – Arrays

(APAS: Quiz > Arrays)

- | | |
|---------------------|----------------|
| 1. absoluteSum1D | 11. symmetry2D |
| 2. find2Max1D | 12. compress2D |
| 3. findMinMax1D | 13. minOfMax2D |
| 4. specialNumbers1D | |
| 5. matVecMult2D | |
| 6. findAverage2D | |
| 7. computeTotal2D | |
| 8. platform1D | |
| 9. swapMinMax1D | |
| 10. diagonal2D | |

Suggested solution can be found at (need VPN is accessing from outside NTU):

<http://172.21.147.174/>NTUQA>





Review Lecture – Week 10

- Week 10 – Learning Materials
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
- **Reviews on Arrays**
- Examples
- Assignment Submission and Grading

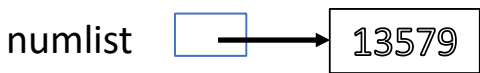


Python vs C – 1D Arrays

	Python	C
list/array initialization	numList = [1, 3, 5, 7, 9]	int numlist[] = {1, 3, 5, 7, 9}; <div style="text-align: right;"> numlist → 13579 </div> numlist – array variable containing a pointer constant [can't be changed] numlist == &numlist[0]
list/array element assignment	numList[2] = 8	numlist[2] = 5; // using index *numlist == numlist[0] == 1; // using pointer *(numlist+2) == numlist[2] == 5; numlist contains a pointer constant , so: numlist += 2; // illegal – cannot update numlist numlist++; // illegal <u>Pointer variable:</u> int *num_ptr; num_ptr = numlist; num_ptr += 2; // legal num_ptr++; // legal <div style="text-align: right; margin-top: 20px;"> numlist → 13579 num_ptr ↗ 13579 </div>








Python vs C – 1D Array Operators

	Python	C
list/array operators	<p><u>Operations</u> concatenate: + repeat: * indexing: [] slicing: [:] membership: in</p> <p><u>List Functions</u> len(), min(), max(), sum()</p> <p><u>List Methods</u> append(), extend(), pop(), insert(), remove(), sort(), reverse()</p> <p><u>Examples</u> numList.append(5) numList.pop()</p>	<p><u>Operations</u> indexing: [] pointer</p> <p><u>Example</u> numlist[2] = 5; *(numlist+2) = 5;</p> <p>Arrays in C has no other comparable array operations.</p> <p>We would need to write our own versions of pop(), append(), etc.</p> 



Python vs C – 1D Array Processing (Examples)

	Python	C
iterating over arrays using indexing	<pre>nums = [1, 2, 3, 4, 5] for i in range(len(nums)): nums[i] += 1</pre>	<pre>int size = 5; int nums[] = {1, 2, 3, 4, 5}; int i; for (i=0; i < size; i++) { nums[i] += 1; }</pre> <div> nums  → 12345 After executing the loop: nums  → 23456 </div>
iterating over arrays using pointers	No real equivalent in Python.	<pre>#include <stdio.h> int main(){ int size = 5, i; int nums[] = {1, 2, 3, 4, 5}; int *p, *last; p = nums; last = p + size; for (p=nums; p < last; p++) *p += 1; printf("nums = "); for (i=0; i<size; i++) printf("%d ",nums[i]); return 0; }</pre> <div> nums  → 12345 p  last  (Arrows point from p and last boxes to the start of the 12345 box) </div> <div> Sample output nums = 2 3 4 5 6 </div> <p>The pointer iteration might be faster, but using indexing is much less error prone.</p>

1D Arrays as Function Arguments in Function Communication

Function Header	<pre>void fn1(int table[], int size) { .. } void fn2(int table[TABLESIZE]) { .. } void fn3(int *table, int size) { .. }</pre> <ul style="list-style-type: none">• table is an array variable (4 bytes)• size and TABLESIZE are the size of the array to be processed.
Calling the Function	<pre>fn1(table, n); // table is 1D array, n is the size of the array table fn2(table); // the defined constant TABLESIZE will be used for the size of the array fn3(table, n);</pre> <ul style="list-style-type: none">• An array table is passed in using call by reference to a function. It means the address of the first element of the array is passed to the function.

Processing 1D Arrays in Functions (Example)

```
#include <stdio.h>
#define SIZE 5
int maximum(int table[SIZE]);
int main()
{
    int max, index, n;
    int numArray[SIZE];
    printf("Enter the number of values: ");
    scanf("%d", &n);
    printf("Enter %d values: ", n);
    for (index = 0; index < n; index++)
        scanf("%d", &numArray[index]);

    max = maximum(numArray);
    printf("The maximum value is %d.\n", max);
    return 0 ;
}
```

numArray

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

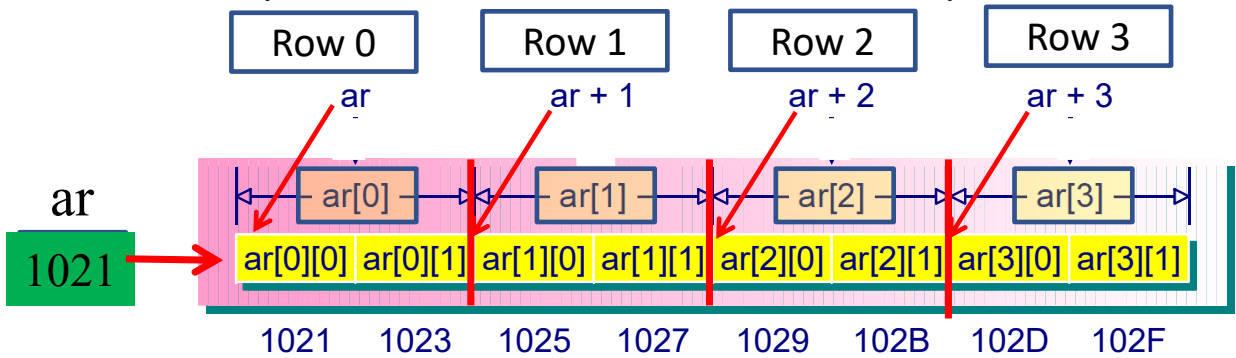
Call by reference

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

// version 2
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;
}
```



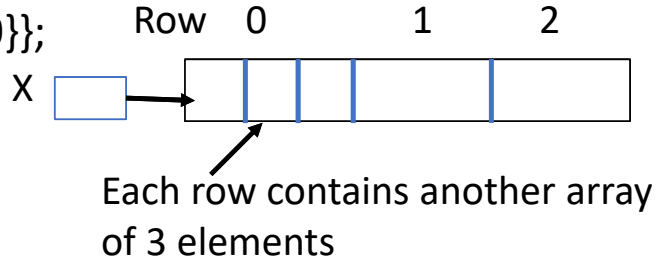
Python vs C – 2D Arrays

	Python	C
2D list/array initialization	<p><code>X = [[1,2,3], [4,5,6], [7,8,9]]</code></p> <p>Nested list – list in list</p>	<p><code>int ar[4][2]={ {1,2}, {3,4}, {5,6},{7,8} };</code></p> <ul style="list-style-type: none"> <code>ar</code> is an array of 4 elements; each element is an array of 2 ints  <p>Formula with pointer (no need to remember):</p> <ul style="list-style-type: none"> <code>ar[m][n] = *(*(ar + m) + n)</code>
2D list/array element assignment	<code>numList[1][1] = 5</code>	<p>(1) <code>ar[m][n]</code>; e.g.: <code>ar[1][1] = 4;</code> // using indexes</p> <p>(2) Using pointer formula for 2D arrays:</p> <p><code>ar[2][1] = *(*(ar + 2) + 1)</code> [m=2, n=1]</p> <p><code>ar[3][0] = *(*(ar + 3) + 0)</code> [m=3, n=0]</p>

Python vs C – Processing 2D Arrays using Indexing (Example)

	Python	C
iterating over 2D arrays using indexing	<pre># Matrix addition – using the list data structure in Python X = [[1,2,3], [4,5,6], [7,8,9]] Y = [[9,8,7], [6,5,4], [3,2,1]] result = [[0,0,0], [0,0,0], [0,0,0]] # iterate through rows and cols for i in range(len(X)): for j in range(len(X[0])): result[i][j] = X[i][j] + Y[i][j] for r in result: print(r)</pre> <p>Sample output</p> <pre>[10 10 10] [10 10 10] [10 10 10]</pre>	<pre>#include <stdio.h> int main() { int X[3][3]={1,2,3},{4,5,6},{7,8,9}}; int Y[3][3]={9,8,7},{6,5,4},{3,2,1}}; int result[3][3] ={{0,0,0},{0,0,0},{0,0,0}}; int i,j; for (i=0; i<3; i++) // using nested loop – traverse row for (j=0; j<3; j++) // traverse column result[i][j] = X[i][j] + Y[i][j]; for (i=0; i<3; i++){ // print 2D array printf("["); for (j=0; j<3; j++){ printf("%d", result[i][j]); if (j!=2) printf(" "); else printf (""); } printf("\n"); } return 0; }</pre> <div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p>Sample output</p> <pre>[10 10 10] [10 10 10] [10 10 10]</pre> </div>

Python vs C – Processing 2D Arrays using Pointers (Example)

	Python	C
iterating over 2D arrays using pointers	No real equivalent in Python.	<pre> /* Matrix addition */ #include <stdio.h> int main() { int X[3][3]={1,2,3},{4,5,6},{7,8,9}}; int Y[3][3]={9,8,7},{6,5,4},{3,2,1}}; int result[3][3] ={{0,0,0},{0,0,0},{0,0,0}}; // double dereferencing via pointers for (i=0; i<3; i++) for (j=0; j<3; j++) *(*result+i)+j) = *(*X+i)+j) + *(*Y+i)+j); } </pre> <p>It is highly recommended to use indexing instead of pointers for processing 2D arrays.</p> <div style="text-align: right;"> <p>E.g. int X[3][3]; /* ar is an array of 3 elements; each element is an array of 3 ints */</p>  </div>

2D Arrays as Arguments in Function Communication

```
#include <stdio.h>
```

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

```
int main(){
```

```
    int A[5][5]={ {1,2,3,4,5},{..}, .. }; // array initialization
```

```
    int i, j, min, max;
```

```
    minMax(A, &min, &max);
```

```
    printf("minMax(): min = %d; max = %d\n", min, max);
```

```
    return 0;
```

```
}
```

```
// Using indexing
```

```
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];
```

```
        }
```

```
}
```

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

```
    int i,*p;
```

```
    p=*a; // assigning the address of array a[0] to p
```

```
    // but if p=&a[0]; may get warning message, still work
```

```
    *max = *p; *min = *p;
```

```
    for (i=0; i<25; i++) {
```

```
        if (*p > *max)
```

```
            *max = *p;
```

```
        else if (*p < *min)
```

```
            *min = *p;
```

```
        p++;
```

```
    }
```

```
}
```

A

a

p

Consecutive & sequential memory



p++

Applying 1D Array to Process 2D Arrays in Functions

```
#include <stdio.h>
void display1(int *ptr, int size);
void display2(int ar[], int size);
int main()
{
    Row: array[0]   array[1]
    int array[2][4] = { 0, 1, 2, 3, 4, 5, 6, 7 };
    int i;
    for (i=0; i<2; i++) {
        display1(array[i], 4);
        display2(array[i], 4);
    }
    display1(array, 8);
    display2(array, 8);
    return 0;
}
```

```
void display1(int *ptr, int size) // using pointer
{
    int j;

    printf("Display1 result: ");
    for (j=0; j<size; j++)
        printf("%d ", *ptr++);
    putchar('\n');
}

void display2(int ar[], int size) // using indexing
{
    int k;
    printf("Display2 result: ");
    for (k=0; k<size; k++)
        printf("%d ", ar[k]*5);
    putchar('\n');
}
```

Use 1D array (a **pointer**) as parameter in a function for processing 2D arrays by passing in the 2D array's starting memory address.

ptr





Review Lecture – Week 10

- Week 10 – Learning Materials
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
- Reviews on Arrays
- **Examples**
- Assignment Submission and Grading





Example 1 – maxNum

Given two lists of grades (list of integers) from two classes, write a program that will check which class has the **highest average score** and the **highest maximum score**.

Program Output:

```
>>>  
Highest Avg:  45.125  
Highest Score: 98  
>>> |
```

Example 1 – Suggested Code

Python: main

```
scores1 = [10, 11, 15, 20, 55, 76, 90, 84]
scores2 = [4, 9, 12, 98, 35, 42, 4, 5, 10]

#avg1 = float(sum(scores1) / len(scores1))
avg1 = sum(scores1) / len(scores1)
avg2 = float(sum(scores2) / len(scores2))

maxNum = max(max(scores1), max(scores2))
maxAvg = max(avg1, avg2)

print("Highest Avg:", "{:.2f}".format(maxAvg))
print("Highest Score:", "{:.2f}".format(maxNum))
```

C: main()

```
#include <stdio.h>
int max1(int scores[], int size);
double max2(double num1, double num2);
int sum(int scores[], int size);
int main()
{
    int scores1[8] = {10, 11, 15, 20, 55, 76, 90, 84};
    int scores2[9] = {4, 9, 12, 98, 35, 42, 4, 5, 10};
    int i, sum1=0, sum2=0, len1=8, len2=9;
    double avg1, avg2, maxNum, maxAvg;

    avg1 = (double)sum(scores1, len1) / len1;
    avg2 = (double)sum(scores2, len2) / len2;
    maxNum = max2((max1(scores1, len1)), max1(scores2, len2));
    maxAvg = max2(avg1, avg2);

    printf("Highest Avg: %.2f\n", maxAvg);
    printf("Highest Score: %.2f\n", maxNum);
    return 0;
}
```



Example 1 – Suggested Code

The other functions are needed to be implemented by ourselves.

```
int max1(int scores[], int size){
    int i, max;
    max=scores[0];
    for (i=1; i<size; i++){
        if (max < scores[i])
            max=scores[i];
    }
    return max;
}

double max2(double num1, double num2){
    if (num1<num2)
        return num2;
    else
        return num1;
}

int sum(int scores[], int size){
    int i, sum=0;
    for (i=0; i<size; i++)
        sum += scores[i];
    return sum;
}
```



Example 2 – swapMinMax1D

Write the C function swapMinMax1D() that takes in an array of integers *ar* and *size* (>1) as parameters, finds the index positions of the largest number and smallest number in the array, swaps the index positions of these two numbers, and passes the array to the calling function via call by reference. For example, if *ar* is {1,2,3,4,5}, then the resultant array *ar* will be {5,2,3,4,1} after executing the function. If there are more than one largest or smallest number in the array, we will swap the **last occurrence** of the largest and smallest numbers. For example, if *ar* is {5,2,1,1,8,9,9}, then the resultant array *ar* will be {5,2,1,9,8,9,1} after executing the function.

The function prototype is:

```
void swapMinMax1D(int ar[], int size);
```

Program input and output

Enter array size:

5

Enter 5 data:

1 2 3 4 5

swapMinMax1D(): 5 2 3 4 1



Example 2 – Suggested Code

Python: main

```
def main():
    arLst = []
    size = int(input("Enter array size: \n"))
    print("Enter {:d} data: ".format(size))
    for i in range(0,size):
        ele = int(input())
        arLst.append(ele)
    print(arLst)
    arLst2 = swapMinMax1D(arLst, size)
    print(arLst2)
    print("swapMinMax1D(): ")
    for i in range(0,size):
        print("{:d}".format(arLst2[i]))

main()
```

C: main()

```
#include <stdio.h>
void swapMinMax1D(int ar[], int size);
int main()
{
    int ar[50],i,size;

    printf("Enter array size: \n");
    scanf("%d", &size);
    printf("Enter %d data: \n", size);
    for (i=0; i<size; i++)
        scanf("%d",ar+i);
    swapMinMax1D(ar, size);
    printf("swapMinMax1D(): ");
    for (i=0; i<size; i++)
        printf("%d ",*(ar+i));
    return 0;
}
```



Example 2 – Suggested Code

Python:

```
def swapMinMax1D(arLst, size):
    j=0
    k=0
    min = arLst[0]
    for i in range(1,size):
        if arLst[i]<=min:
            min = arLst[i]
            j=i
    max = arLst[0]
    for i in range(1,size):
        if arLst[i] >=max:
            max = arLst[i]
            k=i
    arLst[j] = max
    arLst[k] = min
    return arLst
```

C:

```
void swapMinMax1D(int ar[], int size)
{
    int max,min,i=0,j=0,k;

    min = *ar; i=0;
    for (i=1; i<size; i++)
        if ( *(ar+i)<=min ) {
            min=*(ar+i);
            j=i;
        }
    max = *ar; k=0;
    for ( i=1; i<size; i++ ){
        if ( *(ar+i)>=max ) {
            max = *(ar+i);
            k=i;
        }
    }
    *(ar+j) = max;
    *(ar+k) = min;
}
```



Example 3 – diagonals2D

Write a C function that accepts a two-dimensional array of integers `ar`, and the array sizes for the rows and columns as parameters, computes the sum of the elements of the two diagonals, and returns the sums to the calling function through the pointer parameters, `sum1` and `sum2`, using call by reference. For example, if the `rowSize` is 3, `colSize` is 3, and the array `ar` is {1,2,3, 1,1,1, 4,3,2}, then `sum1` is computed as $1+1+2=4$, and `sum2` is $3+1+4=8$.

The function prototype is given as follows:

```
void diagonals2D(int ar[][SIZE], int rowSize, int  
colSize, int *sum1, int *sum2);
```

$\Rightarrow \text{sum1} = 4$

$\Rightarrow \text{sum2} = 8$

Program input and output:

Enter row size of the 2D array:

3

Enter column size of the 2D array:

3

Enter the matrix (3x3):

1 2 3

1 1 1

4 3 2

sum1=4; sum2=8



Example 3 – Suggested Code

**Python:
main**

```
def main():
    lst = []
    arLst = []
    rowSize = int(input("Enter row size of the 2D array: \n"))
    colSize = int(input("Enter column size of the 2D array: \n"))
    print("Enter the matrix ({:d}x{:d}): ".format(rowSize,colSize))
    for i in range(0,rowSize):
        for j in range(0,colSize):
            ele = int(input())
            lst.append(ele)
        arLst.append(lst)
        lst = []
    print(arLst)
    sum1, sum2 = diagonal2D(arLst, rowSize, colSize)
    print("diagonal2D(): ")
    print("sum1 = {:d}, sum2 = {:d}".format(sum1, sum2))
main()
```



Example 3 – Suggested Code

C: main()

```
#include <stdio.h>
#define SIZE 10
void diagonals2D(int ar[][SIZE], int rowSize, int colSize, int *sum1, int *sum2);
int main()
{
    int ar[SIZE][SIZE], rowSize, colSize;
    int i, j, sum1=0, sum2=0;

    printf("Enter row size of the 2D array: \n");
    scanf("%d", &rowSize);
    printf("Enter column size of the 2D array: \n");
    scanf("%d", &colSize);
    printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
    for (i=0; i<rowSize; i++)
        for (j=0; j<colSize; j++)
            scanf("%d", &ar[i][j]);
    diagonals2D(ar, rowSize, colSize, &sum1, &sum2);
    printf("sum1=%d; sum2=%d\n", sum1, sum2);
}
```



Example 3 – Suggested Code

Python:

```
def diagonal2D(ar, rowSize, colSize):
    sum1 = 0
    sum2 = 0
    for i in range(0,rowSize):
        for j in range (0,colSize):
            if (i==j):
                sum1 = sum1+ar[i][j];
    for i in range(0,rowSize):
        for j in range(colSize-1,-1,-1):
            if ((i+j)==colSize-1):
                sum2 = sum2+ar[i][j]
    return sum1, sum2
```

C:

```
void diagonals2D(int ar[][SIZE], int
rowSize, int colSize, int *sum1, int *sum2)
{
    int i,j;
    for (i=0;i<rowSize;i++)
        for (j=0; j<colSize; j++)
            if (i==j)
                *sum1 = *sum1+ar[i][j];
    for (i=0;i<rowSize;i++)
        for (j=colSize-1; j>=0; j--)
            if ((i+j)==colSize-1)
                *sum2 = *sum2+ar[i][j];
}
```



Example 4 – minMax2D

Write a C function minOfMax2D() that takes a two-dimensional array matrix of integers ar, and the array sizes for the rows and columns as parameters. The function returns the minimum of the maximum numbers of each row of the 2-dimensional array ar. For example, if the rowSize is 4, colSize is 4, and ar is {{1,3,5,2}, {2,4,6,8}, {8,6,4,9}, {7,4,3,2}}, then the maximum numbers will be 5, 8, 9 and 7 for rows 0, 1, 2 and 3 respectively, and the minimum of the maximum numbers will be 5.

The prototype of the function is given as follows:

```
int minOfMax2D(int ar[][SIZE], int rowSize, int  
colSize);
```

Program input and output

Enter row size of the 2D array:

4

Enter column size of the 2D array:

4

Enter the matrix (4x4):

1 2 3 4

2 3 4 5

5 6 7 8

8 10 2 4

minOfMax2D(): 4

Example 4 – Suggested Code

Python:
main

```
def main():
    lst = []
    arLst = []
    rowSize = int(input("Enter row size of the 2D array: \n"))
    colSize = int(input("Enter column size of the 2D array: \n"))
    print("Enter the matrix ({:d}x{:d}): ".format(rowSize,colSize))
    for i in range(0,rowSize):
        for j in range(0,colSize):
            ele = int(input())
            lst.append(ele)
        arLst.append(lst)
        lst = []
    print(arLst)
    min = minOfMax2D(arLst, rowSize, colSize)
    print("minOfMax2D(): {:d}".format(min))

main()
```



Example 4 – Suggested Code

C: main()

```
#include <stdio.h>
#define SIZE 10
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize);
int main()
{
    int ar[SIZE][SIZE], rowSize, colSize;
    int i,j,min;

    printf("Enter row size of the 2D array: \n");
    scanf("%d", &rowSize);
    printf("Enter column size of the 2D array: \n");
    scanf("%d", &colSize);
    printf("Enter the matrix (%dx%d): \n", rowSize, colSize);
    for (i=0; i<rowSize; i++)
        for (j=0; j<colSize; j++)
            scanf("%d", &ar[i][j]);
    min=minOfMax2D(ar, rowSize, colSize);
    printf("minOfMax2D(): %d\n", min);
    return 0;
}
```



Example 4 – Suggested Code

Python:

```
def minOfMax2D(ar, rowSize, colSize):
    for row in range(0, rowSize):
        max=ar[row][0]
        for col in range(1, colSize):
            if (ar[row][col]>max):
                max=ar[row][col]
        if (row==0):
            min=max
        elif (max<min):
            min=max
    return min
```

C:

```
int minOfMax2D(int ar[][SIZE], int rowSize, int colSize)
{
    int row,col,max,min;

    for (row=0; row<rowSize; row++)
    {
        max=ar[row][0];
        for (col=1; col<colSize; col++)
            if (ar[row][col]>max)
                max=ar[row][col];
        if (row==0)
            min=max;
        else if (max<min)
            min=max;
    }
    return min;
}
```





Review Lecture – Week 10

- Week 10 – Learning Materials
 - Lectures
 - Lab and Tutorial
 - LAMS MCQ Questions
 - Coding Practice Questions
- Reviews on Arrays
- Examples
- **Assignment Submission and Grading**





Assignment

- **Assignment** (Questions are available in APAS and NTULearn: C Programming > Assignment)
 - 1 Question
 - Submission of code for the assignment via APAS
 - Submission deadline: 14 Nov 2021 (Sunday) 23:59

SC1003 Assignment

The NTU Name Card Holder has a capacity of MAX (e.g., 5 for this program) name cards. You are required to write a NTU Name Card Holder Management Program. The program uses an array of MAX structures for the name card holder.

Each structure should hold:

- nameCardID – It stores the name card identification number.
- personName – It stores the name of the person in the name card.
- companyName – It stores the name of the person's company in the name card.





Assignment Submission and Grading

- Read doc – Assignment Submission and Grading in APAS
(NTULearn: C Programming > Assignments)

ASSIGNMENT CODING, SUBMISSION AND GRADING

You are required to submit your answers for assignment questions to the Automated Programming Assessment System (APAS) for marking and grading. Please follow strictly the following guidelines on submission and grading.

- **Submission Deadline** – Submission deadline will be announced for each assignment.
- **Late Submission** – Late submission **will not** be accepted for grading. If you have any problems on submitting the code using the APAS system, please submit your work to our lab staff Ms Eng Hui Fang (ashfeng@ntu.edu.sg) or the lecturer via email before the deadline. Again, late submission after the deadline will not be accepted for grading. As such, the score for that assignment will be 0. Please remember to submit your work before the submission deadline.
- **Grading Results** – Your submitted assignment code will be graded automatically by the APAS system and the results are available for viewing via the APAS system.





Coding Consideration on Arrays for APAS

- When writing code on processing arrays, you will need to be very careful about the error on “**index-out-of-bound**”.
- It refers to using **index** to access array element that exceeds the size of the array allocated to that variable. For example, if size of array is **array_size**, then the max index will be **array_size-1**. Some students' code may make a mistake to allow the index **larger than array_size-1**.
- So when you develop your code, please double check that the index you use in your code (after updating the index) does not exceed the size of the array (i.e. **array_size-1**).
- For **index-out-of-bound**, it means your code will access the memory location in the system that does not belong to your program. It is very dangerous!!
- Note: when you have such error, **APAS** will execute the code and the running result will be incorrect. But sometimes **Code::Blocks** allows it to run.





Assignment Submission and Grading

- The submitted code will be graded automatically by the **APAS** system. As such, make sure that your program **MUST** be able to run and work on APAS, not just on **Code::Blocks**.
- Your submitted code for assignments will be graded by **APAS**.
- As the compiler used in APAS is different from that used in **Code::Blocks**, your program that can run on **Code::Blocks** may have error when running on **APAS** if not writing the code appropriately.
- The grading is carried out based on program input/output data of hidden test cases using the “**exact string matching**” technique.



Assignment Grading: Request for Review

- Note that the grading will only be done after the submission deadline. Grading results will be returned and available for review.
- **Request for Review** – if you have any queries about the grading results, you may fill up the details based on test cases checking in the Request for Review form (available in NTULearn > Assignments) and submit it together with supporting information before the review deadline.
- **Note that the following errors will not be accepted for review:**
 1. You have to make sure that your program is able to run on APAS. Scores will only be given to programs that can run on **APAS**, not **Code::Blocks**. Request for review based on the program that can only run on **Code::Blocks** will **not** be accepted.
 2. Your program must follow the data requirements given in the question, and the sample test cases on program input and output. It is your responsibility to ensure that. Your errors on program input and output format will not be accepted for review.



Thank you !!!

