4. Array and String

Dr. Ami Tusharkant Choksi
Associate Professor,
Computer Engineering Department,
C.K.Pithawala College of Engineering & Technology, Surat.
2019
ami.choksi@ckpcet.ac.in

4. Array & String: sub topics

Concepts of array, one and two dimensional arrays, declaration and initialization of arrays, string, string storage, Built-in string functions

Total Hours: 5

Module Weightage: 15 %

Table of contents

- Size of data types on Ubuntu64bit version
- Concepts of Array
- Declaration and initialization of 1D array
- Declaration and initialization of 2D array
  - String & its operations
- 2DArray Programs

Size of data types on Ubuntu64bit version

```c
#include <stdio.h>

int main()
{
//print size of each data type available in 'C'
printf("size of each data types in 'C'\n");
printf("char: %d\nint:%d\nfloat:%d\ndouble:%d\n",sizeof(char),
sizeof(int),sizeof(float),sizeof(double));

return 0;
}
/*Output:
size of each data types in 'C'
char: 1
int:4
float:4
double:8
*/
```

Concepts of array

- An array is a collection of single data type values.
- Its size is fixed.
- Declare the array.
  - `Syntax is : datatype arrayname[size];`
  - `int data[100];//`array of type `int` with size=`100`
  - In memory it occupies 100*sizeof(int) bytes
- Initialize the array with elements
  - int a[5]={100,200,300,400,500};
- <---------- indices ---------->

0	1	2	3	4
100	200	300	400	500

- <---------- values ---------->
- array named ‘a’ is having fixed size as 5.

Concepts of array...

- Size of array ‘`a`’ is = `5`
- Its index starts from ‘`0`’.
- Last index of array ‘`a`’ is `size-1`, i.e. 5-1=4.
- It is accessed as `a[0], a[1], a[2], a[3], a[4]`
- Trying to print `a[100]` gives garbage number.

first program of Array

```c
#include <stdio.h>

int main()
{
    //initialize array of type 'int' with size 5
    int a[5]={100,200,300,400,500};
    printf("%d",a[100]);
    return 0;
}

/*Output: -1076519173
*/
```

Iterate array elements

```c
//Iterate Array
#include <stdio.h>
int main(void){
 int a[5]={100,200,300,400,500};
 int i;
 //iterate the array elements
 for(i=0;i<5;i++){
  printf("a[%d] : %d\n",i,a[i]);
 }
 return 0;
}
```

```
/*
Output:
a[0] : 100
a[1] : 200
a[2] : 300
a[3] : 400
a[4] : 500
*/
```

Copy of array1 to array2

```c
#include<stdio.h>
int main(){
int a[5]={1,2,3,4,5};
int a2[6], i;
for(i=0;i<5;i++)
        a2[i]=a[i];
for(i=0;i<5;i++)
printf("a2[%d]:%d\n",i,a2[i]);
}
```

```
/*
Output:
a2[0]:1
a2[1]:2
a2[2]:3
a2[3]:4
a2[4]:5
*/
```

Copy 3 elements from array1 and 3 elements from array2

```
#include<stdio.h>
int main(){
int a[5]={1,2,3,4,5};//3 elements a
int a1[5]={6,7,8,9,10};//3 elements a2
int a2[6];
int i,j;
for(i=0;i<3;i++)
        a2[i]=a[i];
for(j=0;j<3;j++)
        a2[i+j]=a1[j];
```

Copy 3 elements from array1 and 3 elements from array2

```
for(i=0;i<6;i++)
        printf("a2[%d]:%d\n",i,a2[i]);
}
```

```
/*
a2[0]:1
a2[1]:2
a2[2]:3
a2[3]:6
a2[4]:7
a2[5]:8 */
```

Excercise: Concat 2 arrays

Exercise:

```
char a1[]={'h','e','l','l','o'};
char a2[]={'w','o','r','l','d'};

a3 should be
{'h','e','l','l','o','w','o','r','l','d'};
```

Solution: Concat 2 arrays

```
#include<stdio.h>
int main(){
char a1[]={'h','e','l','l','o'};
char a2[]={'w','o','r','l','d'};
char a3[11];
int i,j,len;
for(i=0;i<5;i++)
        a3[i]=a1[i];
for(j=0;j<5;j++)
        a3[i+j]=a2[j];
len=i+j;//array length of a3
```

Solution: Concat 2 arrays

```
for(j=0;j<len;j++)
        printf("%c",a3[j]);
}
```

```
/*
Output:
helloworld
*/
```

String

- String is a collection of characters terminated by a null character, i.e.'\0'
- char c[]="hello";
- or
- char c[]={'h', 'e', 'l', 'l', 'o', '\0'};
- Printing string is using two ways:
  - 1. characterwise
  - 2. whole string is %s

String: Print: two ways

```
//String-print-2 ways
char arr[]={'h','e','l','l','o','\0'}
#include<stdio.h>
int main(){
char arr[]="hello";
int i;
for(i=0;arr[i]!='\0';i++)//characterwise
    printf("%c\n",arr[i]);
printf("String: %s\n",arr);//whole string
}
```

String:Print: two ways

```
/*
h
e
l
l
o
String: hello*/
```

Copy of string1 to string2

```
//Copy of String1 to String2
#include<stdio.h>
int main(){
char a1[]="hello",a2[6];
int i;
for(i=0;a1[i]!='\0';;i++)
        a2[i]=a1[i];
a2[i]='\0';
printf("Copied String:%s\n",a2);
}
```

```
/*
Output:
hello
*/
```

String operations :string.h library

- string.h is a library in ‘C’ language to provide string operations, viz. length of string, concatenation, copy, substring, convert to uppercase/lowercase, etc.
- On Ubuntu, one can see help on terminal using,
- `man 3 strcat`
- `man 3 toupper`

man 3 strcpy

```
STRCPY(3)                  Linux Programmer's Manual                  STRCPY(3)

NAME
       strcpy, strncpy - copy a string

SYNOPSIS
       #include <string.h>

       char *strcpy(char *dest, const char *src);

       char *strncpy(char *dest, const char *src, size_t n);

DESCRIPTION
       The  strcpy()  function  copies the string pointed to by src, including
       the terminating null byte ('\0'), to the buffer  pointed  to  by  dest.
       The  strings  may  not overlap, and the destination string dest must be
       large enough to receive the copy.  Beware  of  buffer  overruns!  (See
       BUGS.)

       The  strncpy()  function is similar, except that at most n bytes of src
       are copied.  Warning: If there is no null byte among the first n  bytes
       of src, the string placed in dest will not be null-terminated.
```

man 3 strcat

```
STRCAT(3)                  Linux Programmer's Manual                 STRCAT(3)

NAME
       strcat, strncat - concatenate two strings

SYNOPSIS
       #include <string.h>

       char *strcat(char *dest, const char *src);

       char *strncat(char *dest, const char *src, size_t n);

DESCRIPTION
       The  strcat() function appends the src string to the dest string, over-
       writing the terminating null byte ('\0') at the end of dest,  and  then
       adds  a  terminating  null  byte.  The strings may not overlap, and the
       dest string must have enough space for the  result.   If  dest  is  not
       large  enough, program behavior is unpredictable; buffer overruns are a
       favorite avenue for attacking secure programs.

       The strncat() function is similar, except that
```

Concatenation of strings with and without strcat()

- **Concatenation using strcat():**
- `strcat(dest,src);`
- **Without using strcat():**

```
int dest_len = strlen(dest);
//copying characters from src to end
of dst string
for (i=0;i<n&&src[i]!='\0';i++)
    dest[dest_len+i] = src[i];
dest[dest_len + i] = '\0';//last is
null character
```

2D array

- 2D array is a collection of same data type values.
- Its size is fixed.
- Declare the array.
  - `Syntax is:datatype arrayname[row][col];`
  - `int data2d[5][5];//`array of type `int` with row=5 and column=`5`
  - In memory it occupies row*column*sizeof(int) bytes, i.e. 5*5*4 = 100 bytes
- array named '`data2d`' is having fixed size of `5x5` int elements.
- It is used to represent matrix of mathematics.

2D array: Initialize and iterate array

- Initialize 2D array:
  - `int a[3][3]={ {11,12,13},{21,22,23},{31,32,33}};`
- Iterate 2D array:

```
for(i=0;i<3;i++){
    printf("\n");
    for(j=0;j<3;j++){
     printf("a[%d] : %d ",i,a[i][j]);
    }
    }
```

2D array program:Initialize and iterate

```c
#include <stdio.h>

int main()
{
    //initialize array of type 'int' with size 3
    int a[3][3]={{11,12,13},{21,22,23},{31,32,33}};
    int i,j;

    printf("size of int and array'a' : %d %d",sizeof(int),sizeof(a));
    //iterate the array elements
    for(i=0;i<3;i++){
    printf("\n");
    for(j=0;j<3;j++){
      printf("a[%d][%d] : %d ",i,j,a[i][j]);
    }
    }
    return 0;
}
```

2D array program:output

```
20    /*Output:
21    size of int and array'a' : 4 36
22    a[0][0] : 11 a[0][1] : 12 a[0][2] : 13
23    a[1][0] : 21 a[1][1] : 22 a[1][2] : 23
24    a[2][0] : 31 a[2][1] : 32 a[2][2] : 33
25    */
```

2D matrix scanning and print

```c
#include <stdio.h>

int main()
{
    //initialize array of type 'int' with size 3x3
    int a[3][3], b[3][3];
    int i,j;

    //Scan array elements
    for(i=0;i<3;i++){
    printf("\n");
    for(j=0;j<3;j++){
     printf("a[%d][%d] :",i,j);
     scanf("%d",&a[i][j]);
    }
    }
```

2D matrix scanning and print

```
17      //Print array elements
18      for(i=0;i<3;i++){
19      printf("\n");
20      for(j=0;j<3;j++){
21       printf("a[%d][%d] : %d ",i,j,a[i][j]);
22      }
23      }
24      return 0;
25  }
```

2D matrix scan and print:output

```
27  /*Output:
28  a[0][0] :1
29  a[0][1] :2
30  a[0][2] :3
31
32  a[1][0] :4
33  a[1][1] :5
34  a[1][2] :6
35
36  a[2][0] :7
37  a[2][1] :8
38  a[2][2] :9
39
40  a[0][0] : 1 a[0][1] : 2 a[0][2] : 3
41  a[1][0] : 4 a[1][1] : 5 a[1][2] : 6
42  a[2][0] : 7 a[2][1] : 8 a[2][2] : 9
43  */
```

2D matrix: Transpose

- The transpose of a matrix is a new matrix whose rows are the columns of the original.
- i.e.

$$A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{bmatrix} \qquad \text{Transpose}(A)=\begin{bmatrix}1 & 4 & 7\\ 2 & 5 & 8\\ 3 & 6 & 9\end{bmatrix}$$

2D matrix: Transpose

```c
#include <stdio.h>

int main()
{
    //initialize array of type 'int' with size 3
    int a[3][3]={{11,12,13},{21,22,23},{31,32,33}};
    int i,j,transpose[3][3];

    //Transpose the array elements
    for(i=0;i<3;i++){
    printf("\n");
    for(j=0;j<3;j++){
      transpose[j][i]=a[i][j];
    }
    }
```

2D matrix: Transpose

```
16      //print original matrix
17      printf("Original Matrix a : \n");
18      for(i=0;i<3;i++){
19      printf("\n");
20      for(j=0;j<3;j++){
21      printf("%d ",a[i][j]);
22      }
23      }
24      //print transpose matrix
25      printf("\nTranspose of Matrix :\n");
26      for(i=0;i<3;i++){
27      printf("\n");
28      for(j=0;j<3;j++){
29      printf("%d ",transpose[i][j]);
30      }
31      }
32      return 0;
33  }
```

2D matrix: Transpose: Output

```
34
35    /*Output:
36    Original Matrix a :
37
38    11 12 13
39    21 22 23
40    31 32 33
41    Transpose of Matrix :
42
43    11 21 31
44    12 22 32
45    13 23 33
46    */
```

Two matrix addition

- Addition of a matrix two matrix is adding both matrix element by element one.
- i.e.

1	2	3
4	5	6
7	8	9

+

10	20	30
40	50	60
70	80	90

=

11	22	33
44	55	66
77	88	99

Two matrix addition

```
1   #include <stdio.h>
2
3   int main()
4   {
5       //initialize array of type 'int' with size 3x3
6       int a[3][3], b[3][3], result[3][3];
7       int i,j;
8
9       //Scan array elements of 'a'
10      for(i=0;i<3;i++){
11      printf("\n");
12      for(j=0;j<3;j++){
13       printf("a[%d][%d] :",i,j);
14       scanf("%d",&a[i][j]);
15      }
16      }
```

Two matrix addition

```
17      //Scan array elements of 'b'
18      for(i=0;i<3;i++){
19      printf("\n");
20      for(j=0;j<3;j++){
21       printf("b[%d][%d] :",i,j);
22       scanf("%d",&b[i][j]);
23      }
24      }
25      //Performing matrix addition and print array elements
26      for(i=0;i<3;i++){
27      printf("\n");
28      for(j=0;j<3;j++){
29       result[i][j]=a[i][j]+b[i][j];
30       printf("result[%d][%d] : %d ",i,j,result[i][j]);
31      }
32      }
33      return 0;
34  }
```

Two matrix addition: Output

```
/*Output:
a[0][0] :1
a[0][1] :2
a[0][2] :3

a[1][0] :4
a[1][1] :5
a[1][2] :6

a[2][0] :7
a[2][1] :8
a[2][2] :9

b[0][0] :10
b[0][1] :20
b[0][2] :30

b[1][0] :40
b[1][1] :50
b[1][2] :60

b[2][0] :70
b[2][1] :80
b[2][2] :90
```

Two matrix addition: Output

```
60
61    result[0][0] : 11 result[0][1] : 22 result[0][2] : 33
62    result[1][0] : 44 result[1][1] : 55 result[1][2] : 66
63    result[2][0] : 77 result[2][1] : 88 result[2][2] : 99*/
```

Two Matrices Subtraction

```
#include <stdio.h>
int main()
{
   int matrix_A[20][20], matrix_B[20][20],
matrix_C[20][20];
    int i,j,row,col;
    scanf("%d",&row); //Accepts number of rows
    scanf("%d",&col); //Accepts number of
columns
```

Two Matrices Subtraction

```
/* Elements of first matrix are accepted from
test data */
    for(i=0; i<row; i++)
    {
        for(j=0; j<col; j++)
        {
            scanf("%d", &matrix_A[i][j]);
        }
    }
```

Two Matrices Subtraction

```
/* Elements of second matrix are accepted from
test data */

    for(i=0; i<row; i++)
    {
        for(j=0; j<col; j++)
        {
            scanf("%d", &matrix_B[i][j]);
        }
    }
```

Two Matrices Subtraction

```
//Subtraction of matrices, A-B
 for(i=0; i<row; i++)
    {
        for(j=0; j<col; j++)
        {

matrix_C[i][j]=matrix_A[i][j]-matrix_B[i][j];
            printf("%d ", matrix_C[i][j]);
        }
   printf("\n");
    }
}
```

Two Matrices Subtraction

```
/*
Output:
If the given matrix are
[A]
2 3 5
4 5 6
6 5 7
[B]
1 5 2
2 3 4
3 3 4
[C]
1 -2 3
2 2 2
3 3 2
The elements of
the output
matrix are
separated by one
blank space*/
```

Lower triangle of a square matrix

```
#include <stdio.h>
int main()
{
int matrix[20][20];
int i,j,r;

scanf("%d", &r); //Accepts number of rows or
columns
```

Lower triangle of a square matrix

```
    for(i=0;i< r;i++) //Accepts the matrix
elements from the test case data
    {
        for(j=0;j<r; j++)
        {
            scanf("%d",&matrix[i][j]);
        }
    }
```

Lower triangle of a square matrix

```
//Lower matrix
    for(i=0;i< r;i++) //Accepts the matrix
elements from the test case data
    {
        for(j=0;j<r; j++){
          if(j<=i)
            printf("%d ",matrix[i][j]);
          else
            printf("0 ");}
      printf("\n");
    }
}
```

Lower triangle of a square matrix

```
/*Output:
For example the output of a given matrix

2 3 4      will be          2 0 0
5 6 7                       5 6 0
4 5 6                       4 5 6
*/
```

Matrix multiplication

- Multiply a matrix by another matrix, we matrix1 if of size MxN and matrix2 is of size NxP
- Resultant matrix will be of size MxP, i.e.
- matrix1 size 2x3, matrix2 size 3x2, result will be 2x2

1	2	3
4	5	6

*

7	8
9	10
11	12

=

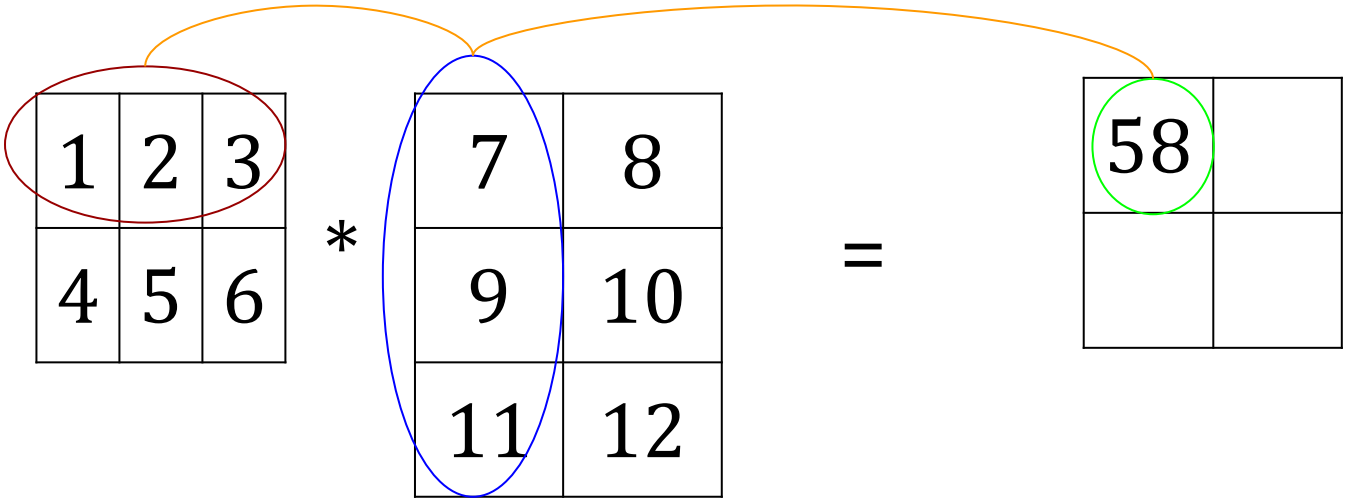
58	64
139	154

Matrix multiplication

- Matrix multiplication is not commutative:
- i.e., $AB \neq BA$

Matrix multiplication

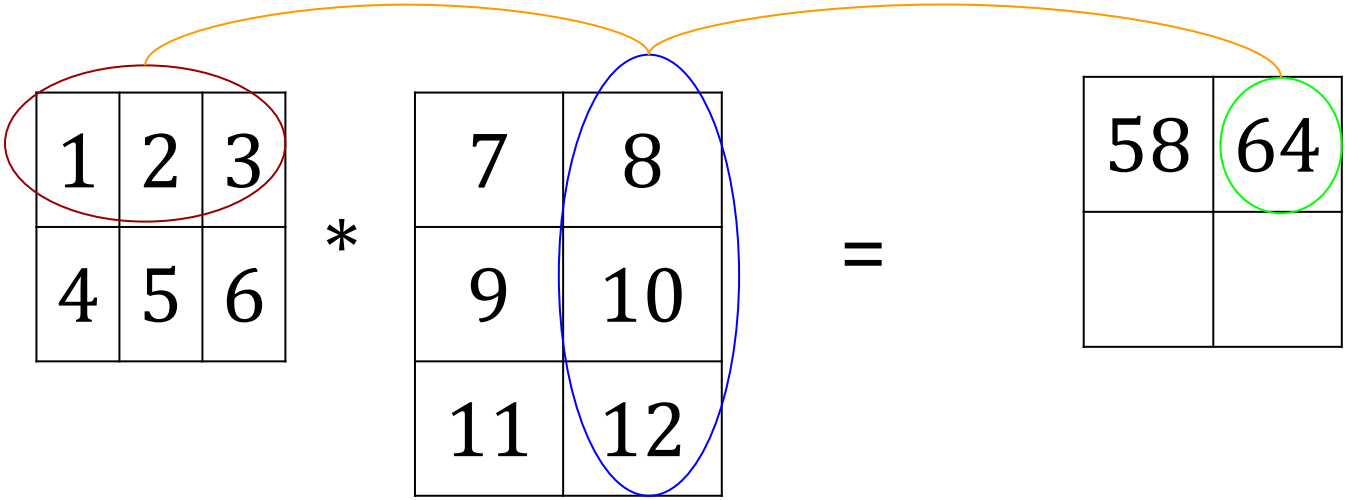
- Multiply a matrix by another matrix we need to do the "dot product" of rows and columns



- 1*7+2*9+3*11=58

Matrix multiplication

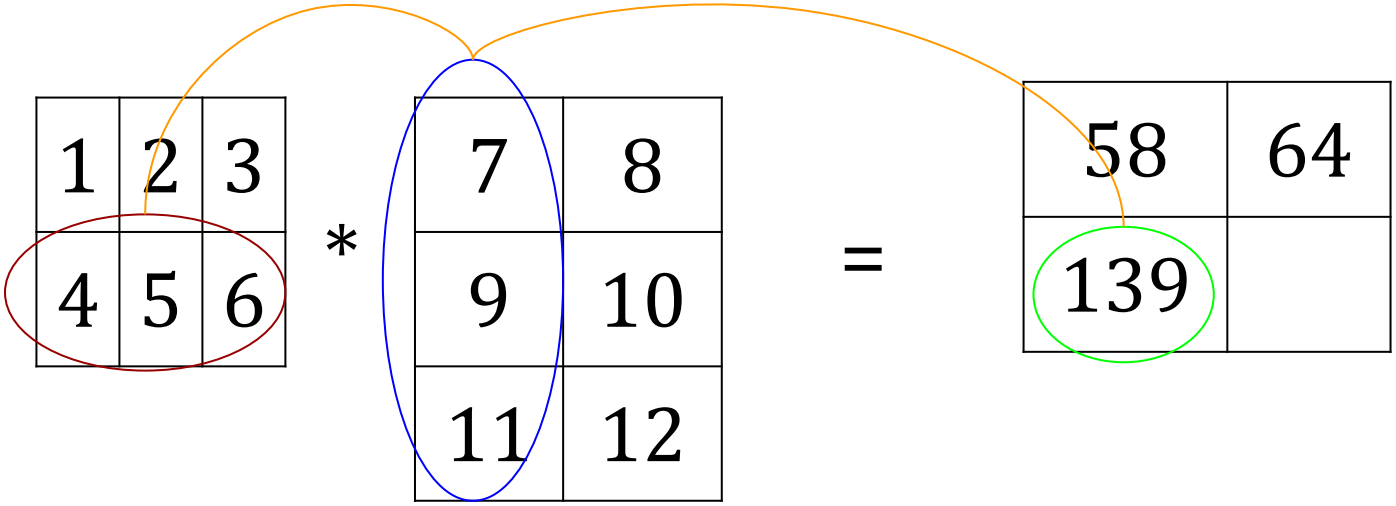
- Multiply a matrix by another matrix we need to do the "dot product" of rows and columns



- 1*8+2*10+3*12=64

Matrix multiplication

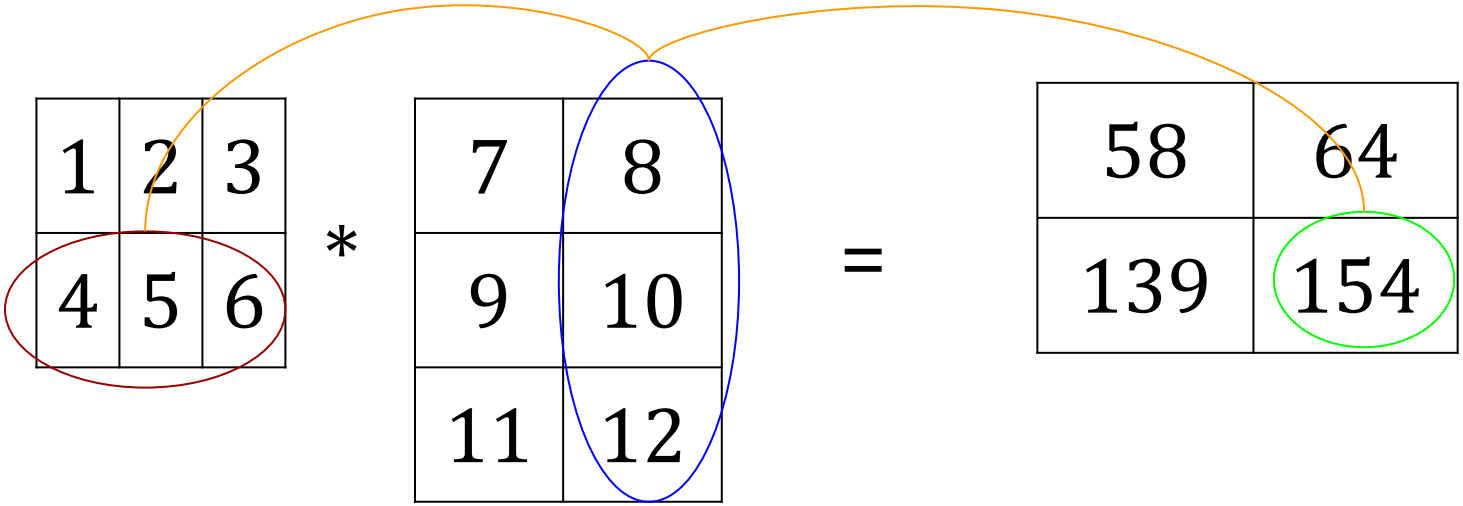
- Multiply a matrix by another matrix we need to do the "dot product" of rows and columns



- 4*7+5*9+6*11=139

Matrix multiplication

- Multiply a matrix by another matrix we need to do the "dot product" of rows and columns



- 4*8+5*10+6*12=154

Real Life example:matrix multiplication

- Orange cost ₹12 each
- Mango cost cost ₹20 each
- Banana cost ₹5 each
- how many they sold in 4 days:

	Mon	Tue	Wed	Thu
Orange	20	30	40	50
Mango	2	10	5	23
Banana	3	8	12	5

Real Life example:matrix multiplication

- The value of sales for Monday is calculated this way:
- Orange value + Mango value + Banana value
- 12*20 + 20*2 + 5*3 = 295

	Mon	Tue	Wed	Thu
Orange	20	30	40	50
Mango	2	10	5	23
Banana	3	8	12	5

Matrix multiplication: Program

```
#include <stdio.h>
int main()
{
     int m=3,n=3,p=3;
    //initialize array of type 'int' with size
3x3
    int a[10][10], b[10][10], result[10][10];
    int i,j,k, sum=0;

    //Scan array elements of 'a'
    for(i=0;i<m;i++){
    for(j=0;j<n;j++){
     printf("a[%d][%d] :",i,j);
```

Matrix multiplication: Program

```
 scanf("%d",&a[i][j]);
}
}
//Scan array elements of 'b'
for(i=0;i<n;i++){
for(j=0;j<p;j++){
 printf("b[%d][%d] :",i,j);
 scanf("%d",&b[i][j]);
}
}
```

Matrix multiplication: Program

```
 //Performing matrix multiplication and print
array elements
    for(i=0;i<m;i++){
    for(j=0;j<p;j++){
      for(k=0;k<n;k++){
        sum+=a[i][k]*b[k][j];
      }
    result[i][j]=sum;
    sum=0;
    }
  }
```

Matrix multiplication: Program

```
 //Print Resultant Matrix
    for(i=0;i<m;i++){
    printf("\n");
    for(j=0;j<p;j++){
     printf("%d ",result[i][j]);
    }
    }
    return 0;
}
```

Matrix multiplication: Program

```
/*
a[0][0] :1
a[0][1] :2
a[0][2] :3
a[1][0] :4
a[1][1] :5
a[1][2] :6
a[2][0] :7
a[2][1] :8
a[2][2] :9
b[0][0] :1
b[0][1] :2
b[0][2] :3
b[1][0] :4
b[1][1] :5
b[1][2] :6
b[2][0] :7
b[2][1] :8
b[2][2] :9
30 36 42
66 81 96
102 126 150
```

Matrix Row and Column sum: Program

```
//Addition of rows and columns of a 2D matrix.
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int arr[3][3]={{1,2,3},{4,5,6},{7,8,9}};
    int i,j;
    int sumRow=0,sumCol=0;
    for(i=0;i<3;i++){
    sumRow=0;
    sumCol=0;
```

Matrix Row and Column sum: Program

```
    for(j=0;j<3;j++){
    sumRow+=arr[i][j];
    sumCol+=arr[j][i];
    }

printf("SumRow%d:%d\nSumCol%d:%d\n",i+1,sumRow
,i+1,sumCol);
    }

    return 0;
}
```

Matrix Row and Column sum: Program

```
/*
Output:
SumRow1:6
SumCol1:12
SumRow2:15
SumCol2:15
SumRow3:24
SumCol3:18
*/
~
```

Largest and smallest Substring

```
#include<stdio.h>
#include<string.h>
int main(){
char str[100]={0},substr[100][100]={0};
//str[100] is for storing the sentence and
substr[50][50] is for storing each word.
scanf("%[^\n]s", str); //Accepts the sentence
from the test case data.
char largest[100], smallest[100];
int i,j,k, small=99999, large=0;
```

Largest and smallest Substring

```
for(i=0,j=0,k=0;str[i]!='\0';i++){
  if(str[i]==' '||str[i]=='.'){
    substr[j][k]='\0';
    if(k>large){
      large=k;
      strcpy(largest,substr[j]);}
    else if (k<small){
      small=k;
      strcpy(smallest,substr[j]);}
    k=0;j++;}
  else{substr[j][k]=str[i];k++;}
}
```

Largest and smallest Substring

```
printf("Largest Word is: %s\nSmallest word is:
%s\n", largest,smallest);
}
```

```
/*
Output:
Problem Solving
in C.
Largest Word is:
Problem
Smallest word
is: C
*/
```

Location, noOftimes, notFound key

Write a program to print all the locations at which a particular element(taken as input) is found in a list and also print the total number of times it occurs in the list. The location starts from 1.

Location, noOftimes, notFound key

```
#include <stdio.h>
int main(){
   int array[100], search, n, count = 0;
   //"search" is the key element to search
and 'n' is the total number of element of
the array
   // "count" is to store total number of
elements
scanf("%d", &n); //Number of elements is
taken from test case
```

Location, noOftimes, notFound key

```
    for (c = 0; c < n; c++)
       scanf("%d", &array[c]);
    scanf("%d", &search); // The element to
search is taken from test case
for (c = 0; c < n; c++){
  if(array[c]==search){
   printf("%d is present at location
%d.\n",search,c+1);
     count++;  }
}
```

Location, noOftimes, notFound key

```
if(count ==0)
   printf("%d is not present in the
array.\n",search);
else
   printf("%d is present %d times in the
array.\n",search,count);
}
```

Output:

/*Output:

For example if there are

4 elements in the array

5

6

5

7

If the element to search is 5 then the output will be

5 is present at location 1

5 is present at location 3

5 is present 2 times in the array.

*/

Reverse Array by Swapping elements

Write a C program to reverse an array by swapping the elements and without using any new array.

Reverse Array by Swapping elements

```
#include <stdio.h>
int main() {
  int array[100], n, c;
  int t,j;
  scanf("%d", &n); // n is number of
elements in the array.
  for (c = 0; c < n; c++) {
    scanf("%d", &array[c]);
  }
```

Reverse Array by Swapping elements

```
for (c = 0,j=n-1; c < n/2; c++,j--) {
  t=array[c];
  array[c]=array[j];
  array[j]=t;
}
printf("Reversed array elements are:\n");
  for (c = 0; c < n; c++) {
    printf("%d\n", array[c]);
  }
  return 0;}
```

Reverse Array by Swapping elements

```
/*Output:
5
10
20
30
40
50
Reversed array
elements are:
50
40
30
20
10
*/
```

Merge two sorted arrays

```
#include <stdio.h>
void merge(int a[], int m, int b[], int n, int sorted[]);
int main()
{
int a[100], b[100], m, n, c, sorted[200];
/* a[100] and b[100] are the two given arrays and m and n are the their respective sizes. c is a counter and sorted[200] is the final sorted array */
scanf("%d", &m); //Number of elements in the first array
```

Merge two sorted arrays

```
for (c = 0; c < m; c++)
  {
  scanf("%d", &a[c]); //Elements of first array is read
  }
 scanf("%d", &n); //Number of elements in second array
 for (c = 0; c < n; c++)   {
  scanf("%d", &b[c]); //Elements of second array is read
 }
merge(a, m, b, n, sorted);
```

Merge two sorted arrays

```
//The merged function is called where the two arrays are merged and sorted.


  printf("Sorted array:\n");


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

Merge two sorted arrays

```
void merge(int a[], int m, int b[],
int n, int sorted[]){
  int i, j, k;
  j = k = 0;
  for (i = 0; i < m + n;) {
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;      }
      i++;    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
        i++;      }
    }
```

Merge two sorted arrays

```
else {
    for (; i < m + n;) {
      sorted[i] = a[j];
      j++;
      i++;
    }
  }
 }
}
```

```
/*Output:
5
10
20
30
40
50
6
100
200
300
400
500
600
```

```
Sorted array:
10
20
30
40
50
100
200
300
400
500
 600*/
```

Binary search:iterative

```
#include <stdio.h>
int main(){
 int c, n, search,
 array[100];
 scanf("%d",&n); //number of elements in the array
 for (c = 0; c < n; c++)
 scanf("%d",&array[c]);

 scanf("%d", &search); //The element to search is read from
test case.
```

Binary search:iterative

```
/* Use the printf statements as below:
 printf("%d found at location %d.\n", search, variable_name);
 printf("Not found! %d isn't present in the list.\n", search);
*/
int x=search;
int N=n;
int position=-1;
// search space is A[low..high]
      int low = 0, high = N - 1;
```

Binary search:iterative

```
// iterate till search space contains at-least one element
while (low <= high)   {
      // find the mid value in the search space and
      // compares it with target value
      int mid = (low + high)/2;      // overflow can happen
      // int mid = low + (high - low)/2;
      // int mid = high - (high - low)/2;
      // target value is found
if (x == array[mid]){
 position= mid; break;}
```

Binary search:iterative

```
        // if target is less than the mid element, discard all elements
        // in the right search space including the mid element
    else if (x < array[mid]){
      high = mid - 1;
    }
        // if target is more than the mid element, discard all elements
        // in the left search space including the mid element
```

Binary search:iterative

```
            else
                low = mid + 1;}
if(position!=-1)
     printf("%d found at location %d.\n", search, position+1);
else
 printf("Not found! %d isn't present in the list.\n", search);
}
```

Binary search:iterative

```
/*Output:
5
5 found at location 5
6
 Not found! 6 isn't present in the list.
*/
```

References

1. Array in C, https://www.geeksforgeeks.org/arrays-in-c-cpp/
2. Array in C, https://www.tutorialspoint.com/cprogramming/c_arrays.htm
3. Array in C, https://www.programiz.com/c-programming/c-arrays
4. Transpose of a matrix, https://chortle.ccsu.edu/vectorlessons/vmch13/vmch13_14.html
5. Matrix multiplication, https://www.mathsisfun.com/algebra/matrix-multiplying.html
6. Contents here are also taken from the Assignments I solved while attending NPTEL course titled “Problem solving through Programming In C”, 2019.