Arrays & Strings

Session Objectives

- To learn about array declaration and manipulation
- To learn about matrix operation using two dimensional arrays
- To learn about string handling using arrays.

Session Topics

- Accessing the array elements and array initialization
- Single and multidimensional array
- Strings and string variables: Reading and Printing a string
- Functions and arrays

Arrays

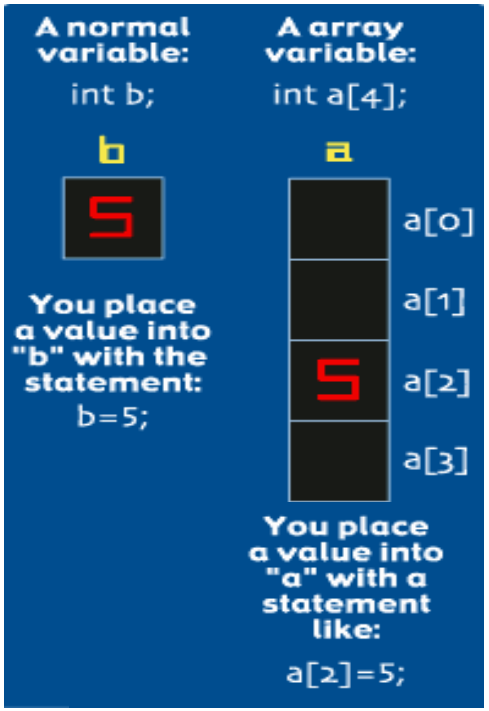
- An array lets you declare and work with a collection of values of the same type. For example, you might want to create a collection of five integers.

  Example :int a, b, c, d, e;
- Using arrays it could be declared as below.

  int a[5];

  This reserves memory for an array that to hold five integer values
- The five separate integers inside this array are accessed by an index. All arrays start at index 0 and go to (n-1) in C. For example:

```
int a[5];
a[0] = 12;
a[1] = 9;
a[2] = 14;
a[3] = 5;
a[4] = 1;
```



- You need to make a declaration before using an array and to specify its data type, its name and, in most cases
- Make sure the array has a valid name

Arrays

- Arrays are storage structures in memory that can help store multiple data items having a common characteristic.
- An array is referred to only by a single name, although they have multiple data items
- The individual data items can be of any *type*, like integers, float or characters.
- But inside one array, all the data items must be of a single type
  - If we have an array of 50 elements, all of them must be of integer type or of any one type
  - We cannot have the first 20 being integers while the rest are float type
- Character arrays have a special property...
- Each element of the array can hold one character. But if you end the array with the NULL CHARACTER, denoted by '\0' (that is, backslash and zero), you'll have what is known as a STRING CONSTANT. The null character marks the end of a string

Defining an array

Generally,

```
storage_class     data_type     array[expression];
```

```
int x[100];
float value[500];
static char message[10];
static float k[10];
extern int m[500];
```

Initializing Arrays

- You can assign values to the array in several ways
- this example demonstrates

```
  int main() {
int arrayOfInts1[8] = {1, 2, 3, 4, 5, 6, 7, 8};
int arrayOfInts2[8];
int arrayOfInts3[ ] = {1,2,3,4,5,6,7,8}; /* an unsized array */
int arrayOfInts4[10];
int i; arrayOfInts2[0] = 1;
arrayOfInts2[1] = 2;
arrayOfInts2[2] = 3;
arrayOfInts2[3] = 4;
arrayOfInts2[4] = 5;
arrayOfInts2[5] = 6;
arrayOfInts2[6] = 7;
arrayOfInts2[7] = 8;
for(i=0 ; i<8 ; i++)
arrayOfInts4[i] = i + 1;
return 0;
}
```

Arrays

- One of the feature about array indexing is that, you can use a loop to manipulate the index. For example, the following code initializes all of the values in the array to 0:

```
int a[5];
int i;
for (i=0; i<5; i++)
        a[i] = 0;
```

- The following code initializes the values in the array sequentially and then prints them out:

```
#include <stdio.h>
    int main()
    {
        int a[5], i;
        for (i=0; i<5; i++)
          a[i] = i;
        for (i=0; i<5; i++)
           printf("a[%d] = %d\n", i, a[i]);
    }
```

Example : READ and Display

```
#include <stdio.h>
    int main()
    {
    int a[5], i;

    for (i=0; i<5; i++)
            a[i] = i;

    for (i=0; i<5; i++)
            printf("a[%d] = %d\n", i, a[i]);
    }
```

```
#include <stdio.h>
    int main()
    {
    int a[5], i;

    for (i=0; i<5; i++)
            scanf("%d",&a[i]);

    for (i=0; i<5; i++)
            printf("a[%d] = %d\n", i, a[i]);
    }
```

Example : READ, Process and Display

```
#include<stdio.h>
void main( )
{
int n,a[100];
printf("How many numbers");
scanf("%d",&n);
int i;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
{
if(i%2==0)
{
a[i]=a[i]+5;
}
else
a[i]=a[i]*a[i];
}
for(i=0;i<n;i++)
printf("a[%d]=%d\n",i,a[i]);
}
```

Example: Read, Process and Display

```
void main( )
{
int n,a[100];
printf("How many numbers");
scanf("%d",&n);
int i;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
{

a[i]=a[i]+5;
}
for(i=0;i<n;i++)
printf("a[%d]=%d\n",i,a[i]);
}
```

```
void main( )
{
int n,a[100];
printf("How many numbers");
scanf("%d",&n);
int i;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n;i++)
{
a[i]=0;
}
for(i=0;i<n;i++)
printf("a[%d]=%d\n",i,a[i]);
}
```

Example 2: calculation of Mean

```
int main( )
{
int a[100], i,n;
float sum=0,mean;
printf("How many numbers");
scanf("%d",&n);
for (i=0; i < n; i++)
        scanf("%d",&a[i]);
for (i=0; i <n; i++)
        sum=sum+a[i];
mean=sum/n;
printf("Mean of the given numbers
   is:%f",mean);
}
```

```
int main( )
{
int s,i,n;
float sum=0,mean;
printf("How many numbers");
scanf("%d",&n);
for (i=0; i < n; i++)
   {
   scanf("%d",&a);
   sum=sum+a;
   }
mean=sum/n;
printf("Mean of the given numbers
   is:%f",mean);
}
```

Example 3: Calculation of Deviation

- Modify example 2 so that it will be able to calculate deviation of each number from mean…

? ? ?

```
int main( )
{
int a[100], i,n;
float sum=0,mean,devi[100];
printf("How many numbers");
scanf("%d",&n);
for (i=0; i < n; i++)
        scanf("%d",&a[i]);
for (i=0; i < n; i++)
        sum=sum+a[i];
mean=sum/n;
for( i=0; i < n; i++)
devi[i] = a[i]-mean;
for(i=0; i < n; i++)
printf("deviation of given numbers a[%d] from original num is
   dev[%f]\n",a[i],devi[i]);
}
```

Example 4: calculation of S.D.

- Calculate the standard deviation of given numbers using following formula

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2},$$

Minimum Calculation

```
#include <stdio.h>
#define size 10
main( )
{
int a[size],i, min;
printf("Give 10 values \n");
for (i=0; i<size; i++)
scanf("%d", &a[i]);
min = 99999;
for (i=0; i<size; i++)
{
if (a[i] < min)
min = a[i];
}
printf("\n Minimum is %d", min);
}
```

GPA Calculation

```
#include <stdio.h>
#define nsub 6
main()
{
int grade_pt[nsub], cred[nsub],i, gp_sum=0, cred_sum=0, gpa;
printf("Input gr. points and credits for six subjects \n");
for (i=0; i<nsub; i++)
scanf("%d %d", &grade_pt[i], &cred[i]);
for (i=0; i<nsub; i++)
{
gp_sum  +=  grade_pt[i] * cred[i];
cred_sum += cred[i];
}
gpa= gp_sum / cred_sum;
printf("\n Grade point average: is %d", gpa);
}
```

Passing Array to a function

An array name can be used as an argument to a function.

–Permits the entire array to be passed to the function.

–Array name is passed as the parameter, which is effectively the address of the first element.

Rules:

–The array name must appear by itself as argument, without brackets or subscripts.

–The corresponding formal argument is written in the same manner.

- **Declared by writing the array name with a pair of empty brackets.**
- **Dimension or required number of elements to be passed as a separate parameter.**

Example: Average of numbers

```
#include <stdio.h>
float avg(float [], int);
main()
{
float a[]={4.0, 5.0, 6.0, 7.0};
printf("%f \n“,avg(a,4));
}
```

```
float avg(float x[], int n)
{
float sum=0;inti;
for(i=0; i<n; i++)
sum+=x[i];
return(sum/(float) n);
}
```

The Actual Mechanism

- **When an array is passed to a function, the values of the array elements are not passed to the function.**
  - **The array name is interpreted as the address of the first array element.**
  - **The formal argument therefore becomes a pointer to the first array element.**
  - **When an array element is accessed inside the function, the address is calculated using the formula stated before.**
  - **Changes made inside the function are thus also reflected in the calling program.**
- **Passing parameters in this way is called call-by-reference.**
- **Normally parameters are passed in C using**
  - **call-by-value.**
- **Basically what it means?**
  - **If a function changes the values of array elements, then these changes will be made to the original array that is passed to the function.**
  - **This does not apply when an individual element is passed on as argument.**

Example: Minimum from a set of numbers

```
#include <stdio.h>
main( )
{
int a[100], i, n;
scanf("%d", &n);
for (i=0; i<n; i++)
scanf("%d", &a[i]);
printf("\n Minimum is %d",
   minimum (a, n));
}
```

```
int minimum (x[ ], size)
{
int i, min = 99999;
for (i=0; i<size; i++)
if (min < a[i])
min = a[i];
return (min);
}
```

Example : Multiple functions

```
#include<stdio.h>
int read(int n,int a[ ]);
int write(int n,int a[ ]);
int process(int n,int a[ ]);
void main()
{
int n,b[100];
printf("How many numbers");
scanf("%d",&n);
read(n,b);
process(n,b);
write(n,b);
}
```

```
int read(int n,int a[ ])
{
int i;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
return;
}

int process(int n,int a[ ])
{
int i;
for(i=0;i<n;i++)
a[i]=a[i]+5;
return;
}
```

```
int write(int n,int a[ ])
{
int i;
for(i=0;i<n;i++)
printf("a[%d]=%d\n",i,a[i]);
return;
}
```

Example: Standard deviation using 2 funcions

```
#include<stdio.h>
#include<math.h>
int read(int n,int a[ ]);
int process(int n,int a[ ]);

void main()
{
int n,b[100];
printf("How many numbers");
scanf("%d",&n);
read(n,b);
process(n,b);
}
```

```
int read(int n,int a[ ])
{
int i;
for(i=0;i<n;i++)
scanf("%d",&a[i]);
return;
}

int process(int n,int a[])
{
int i;
float d[100],v[100];
float sum=0,mean,sum1=0,sd;
for(i=0;i<n;i++)
sum=sum+a[i];
mean=sum/n;
```

```
for(i=0;i<n;i++)
d[i]=a[i]-mean;
for(i=0;i<n;i++)
v[i]=pow(d[i],2);
for(i=0;i<n;i++)
sum1=sum1+v[i];
sd=sqrt(sum1/(n-1));
printf("standard
deviation=%f",sd);
return;
}
```

Sorting an array

```
#define size 100
int reorder (int n,int x[ ]);
void main( )
{
int i,n,x[size];
printf("how many numebers\n");
scanf("%d",&n);
for(i=0;i<n;i++)
{
    printf("i=%d    x= ",i+1);
    scanf("%d",&x[i]);
 }
reorder(n,x);
for(i=0;i<n;i++)
printf("i=%d    x[%d] =%d", i,i,x[i]);
}
```

```
int reorder (int n,int x[])
{
int i,j,temp;
for( i=0;i < n-1; i++)
{
for( j=i+1;j < n ; j++)
if(x[j] < x[i])
{
    temp=x[i];
    x[i] = x[j];
    x[j]=temp;
}
}
return;
}
```

Multi-Dimensional Arrays

- An array's ***DIMENSION*** is the number of indices required to reference an element.
- For example, arrayOfInts[0] is the first element. The index is 0 - there is only 1 index so arrayOfInts is one dimensional.
- what about 2D?

```
int array2D[3][5];
```

This tells the computer to reserve enough memory space for an array with 15, that is, 3 x 5 elements.

- If you ever wanted to find out how much memory your arrays occupy (1D or multidimensional), you can use the sizeof() operator.

2-D Array

- **We have seen that an array variable can store a list of values.**
  - **Many applications require us to store a table of values.**

	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5
Student 1	75	82	90	65	76
Student 2	68	75	80	70	72
Student 3	88	74	85	76	80
Student 4	50	65	68	40	70

- **The table contains a total of 20 values, five in each line.**
  - **–The table can be regarded as a matrix consisting of four rows and five columns.**
  - **- C allows us to define such tables of items by using two-dimensional arrays.**

Declaring 2-D Arrays

General form:

```
storage_class data_type array_name [row_size][column_size];
```

•Examples:

```
int marks[4][5];
float sales[12][25];
double matrix[100][100];
static double matrix[100][100];
```

•Accessing Elements of a 2-D Array

- **Similar to that for 1-D array, but use two indices.**
  - **–First indicates row, second indicates column.**
  - **- Both the indices should be expressions which evaluate to integer values.**

•Examples:

```
x[m][n] = 0;
c[i][k] += a[i][j] * b[j][k];
```

Starting from a given memory location, the elements are stored row-wise in consecutive memory locations.
x: starting address of the array in memory
c: number of columns
k: number of bytes allocated per array element

a[i][j] ➡ is allocated memory location at address

$$x + (i * c + j) * k$$

a[0][0] a[0][1] a[0][2] a[0][3] a[1][0] a[1][1] a[1][2] a[1][3] a[2][0] a[2][1] a[2][2] a[2][3]

How to read the elements of a 2-D array?

By reading them one element at a time

```
for (i=0; i<nrow; i++)
for (j=0; j<ncol; j++)
        scanf ("%f", &a[i][j]);
```

The ampersand (&) is necessary. The elements can be entered all in one line or in different lines.

2-D Array (contd...)

By printing them one element at a time.

```
for (i=0; i<nrow; i++)
for (j=0; j<ncol; j++)
printf ("\n %f", a[i][j]);
```

–The elements are printed one per line.

```
for (i=0; i<nrow; i++)
{
printf ("\n");
for (j=0; j<ncol; j++)
printf("%f ", a[i][j]);
}
```

2-D array Initialization

```
int x[3] [4] = {1,2,3,4,5,6,7,8,9,19,-5,5};
int x[3] [4] = {1,2,3,4,5,6,7};
int x[3] [4] = {
                {1,2,3,4},
                {5,6,7,8},
                 {9,10,11,12}
              };
int x[3] [4] = {
                {1,2,3},
                {5,6,7},
                 {9,10,11}
              };
int x[3] [4] = {
                {1,2,3,4,5},   /* Error */
                {5,6,7,8,6},
                 {9,10,11,12,13}
              };
```

Example :Read and Display

```
#include<stdio.h>
main()
{
int a[100][100], p, q, m, n;
scanf("%d %d", &m, &n);
for (p=0; p<m; p++)
for (q=0; q<n; q++)
scanf("%d", &a[p][q]);

for (p=0; p<m; p++)
{
printf ("\n");
for (q=0; q<n; q++)
printf("%d ", a[p][q]);
}
}
```

Example :Read, Process and Display

```
#include<stdio.h>
main()
{
int a[100][100], p, q, m, n;
scanf("%d %d", &m, &n);
for (p=0; p<m; p++)
for (q=0; q<n; q++)
scanf("%d", &a[p][q]);

for (p=0; p<m; p++)
for (q=0; q<n; q++)
a[p][q]=a[p][q]+5;

for (p=0; p<m; p++)
{
printf ("\n");
for (q=0; q<n; q++)
printf("%d ", a[p][q]);
}
}
```

Example : Matrix Addition

```
#include<stdio.h>
main()
{
int a[100][100],
b[100][100],c[100][100], p, q, m, n;
scanf("%d %d", &m, &n);
for (p=0; p<m; p++)
for (q=0; q<n; q++)
scanf("%d", &a[p][q]);
for (p=0; p<m; p++)
for (q=0; q<n; q++)
scanf("%d", &b[p][q]);
for (p=0; p<m; p++)
for (q=0; q<n; q++)
c[p]q] = a[p][q] + b[p][q];
for (p=0; p<m; p++)
{
printf ("\n");
for (q=0; q<n; q++)
printf("%f ", a[p][q]);
}
}
```

Modify this program such that program will take row1,colum1 and row2,column 2 from user. Check whether addition is possible or not.

Passing 2-D array to Function

Similar to that for 1-D arrays.

–The array contents are not copied into the function.

–Rather, the address of the first element is passed.

•For calculating the address of an element in a 2-D array, we need:

–The starting address of the array in memory.

–Number of bytes per element.

–Number of columns in the array.

•The above three pieces of information must be known to the function.

```
#include <stdio.h>

main()
{
   int a[15][25], b[15]25];
   :
   :
   add (a, b, 15, 25);
   :
}
```

```
void add (x, y, rows, cols)
int x[][25], y[][25];
int rows, cols;
{
   :
}
```

We can also write

int x[15][25], y[15][25];

Number of columns

Example: Transpose

```
void  transpose (int x[][100], n)
{
   int  p, q;

   for  (p=0; p<n; p++)
      for  (q=p; q<n; q++)
        {
            t = x[p][q];
            x[p][q] = x[q][p];
            x[q][p] = t;
        }
}
```

10	20	30
40	50	60
70	80	90

⇩

10	40	70
20	50	80
30	60	90

Try it!!!

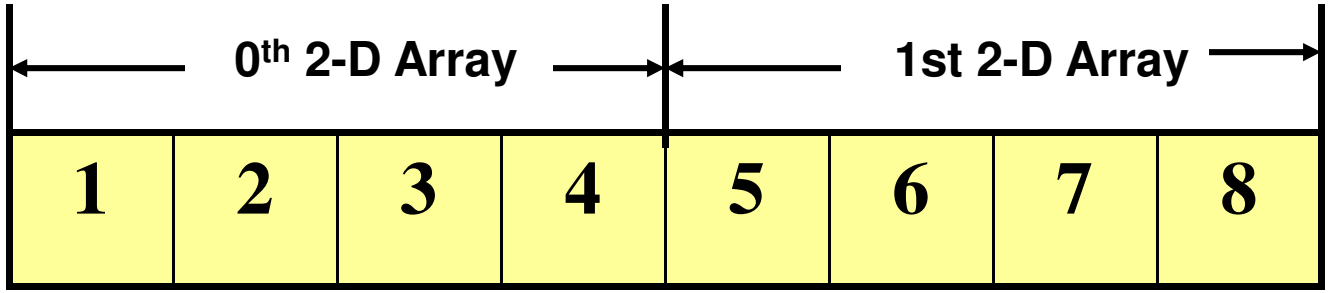
1. **WAP to insert 12 elements in a 3*4 matrix and display those values.**
   A. **Modify q.no.1 so that all the elements will be zero and display the value.**
   B. **Add 10 if the elements are diagonal elements otherwise multiply by the previous value stored in matrix for q.no.1.**
   C. **Use function to solve problem no. 1,1(A) and 1(B)**
2. **WAP to find matrix multiplication using function.**

3D Array

- A three dimensional array can be thought of as an array of arrays of arrays.
- Example:

```
static int [2][2][2]={
                    {
                            {1,2},
                            {3,4}
                    }
                    {
                            {5,6},
                            {7,8}
                    }
                  }
```

Memory Allocation of a 3D Array



An Introduction to Strings

- Strings in C are an array of charecters terminated with a null character, '\0',.
- This means that the length of a string is the number of characters it contains plus one to store the null character.
- Examples:

```
char     string_1 = "Hello";
char string_2[ ]  = "Hello";
char string_3[6] = "Hello";
```

One can use the string format specifier, %s, to handle strings.

Reading Strings

- One possible way to read in a string is by using scanf. However, the problem with this, is that if you were to enter a string which contains one or more spaces, scanf would finish reading when it reaches a space, or if return is pressed.
- We could use the ***gets*** function...

  gets takes just one argument - a char pointer, or the name of a char array, but don't forget to declare the array / pointer variable first!
- What's more, is that it automatically prints out a newline character, making the output a little neater

Writing Strings

- Like with printf and scanf, if you want to use gets( ) and puts( ) you'd have to include the stdio.h header file.
- puts( ) is similar to gets in the way that it takes one argument - a char pointer. This also automatically adds a newline character after printing out the string.

Strings: An Example

```
#include <stdio.h>
void main()
{
     char array1[50],array2[];
     printf("Now enter another string less than 50");
     printf(" characters with spaces: \n");
     gets(array1);
     printf("\nYou entered: ");
     puts(array1);
     printf("\nTry entering a string less than 50");
     printf(" characters, with spaces: \n");
     scanf("%s", array2);
     printf("\nYou entered: %s\n", array2);
 }
```

Output

```
Now enter another string less than 50 characters with spaces:
hello world

You entered: hello world

Try entering a string less than 50 characters, with spaces:
hello world

You entered: hello
```

String Functions defined in ***string.h***

- strlen(str)---Returns length of the string *str*.
- strcpy(str1,str2)---Copies the string *str2* to string *str1*.
- strncpy(str1,str2,n)---Copies at most *n* charecters of string *str2* to string *str1*.
- strcat(str1,str2)---Append string *str2* to string *str1*.
- strncat(str1,str2,n)---Append first *n* charecters of string *str2* in string *str1*.
- strcmp(str1,str2)---Compare two strings *str1* and *str2*.

String Functions defined in ***string.h***

- strstr(str1,str2)---Finds the occurrence of string *str2* in string *str1*.
- Strset(str1,c)---Sets all occurrence in *str1* to the character identified by ***c.***
- strchr(str,c)---Scans the string *str* for the first occurrence of character ***c***.
- strrchr(str,c)---Find the last occurrence of the character ***c*** in string *str*.
- strlwr(str)---Converts the string *str* to lowercase.
- strupr(str)---Converts the string *str* to uppercase.
- strrev(str)---Reverses the string *str*.

Thank You!