Module 3 : Arrays and Pointers

Arrays

- **Array Declaration**

  int marks[30] ;

- **Accessing Elements of an Array -** is done with subscript, the number in the brackets following the array name. This number specifies the element's position in the array. All the array elements are numbered, starting with 0. Thus, marks[2] is not the second element of the array, but the third

- **Entering Data into an Array**

```
for ( i = 0 ; i <= 29 ; i++ )
{
printf ( "Enter marks " ) ;
scanf ( "%d", &marks[ i ] ) ;
}
```

Arrays

- **Reading Data from an Array**

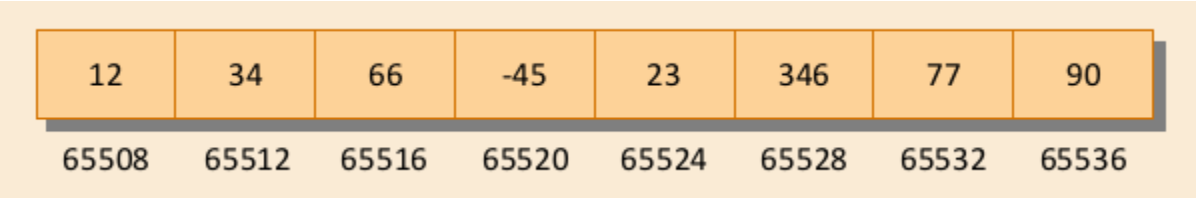
```
for ( i = 0 ; i <= 29 ; i++ )
        sum = sum + marks[ i ] ;
```

- **Array Initialization**

```
int num[ 6 ] = { 2, 4, 12, 5, 45, 5 } ;
int n[ ] = { 2, 4, 12, 5, 45, 5 } ;
float press[ ] = { 12.3, 34.2, -23.4, -11.3 } ;
```

- **Array Elements in Memory**

```
int a[]={12,34,66,-45,23,346,77,90};
```



Arrays

```
# include <stdio.h>
int main( )
{
int num[ 40 ], i ;
for ( i = 0 ; i <= 100 ; i++ )
num[ i ] = i ;
return 0 ;
}
```

No Complilation Error but

*** stack smashing detected ***: terminated

Aborted (core dumped)

Pointers and Arrays

```c
# include <stdio.h>
int main( )
{
int arr[ ] = { 10, 20, 30, 45, 67, 56, 74 } ;
int *i, *j ;
i = &arr[ 1 ] ;
j = &arr[ 5 ] ;
printf ( "%d %d\n", j - i, *j - *i ) ;
return 0 ;
}
```

OUTPUT :

4 36

Comparison of two pointer variables

```c
# include <stdio.h>
int main( )
{
int arr[ ] = { 10, 20, 36, 72, 45, 36 } ;
int *j, *k ;
j = &arr [ 4 ] ;
k = ( arr + 4 ) ;
if ( j == k )
printf ( "The two pointers point to the same location\n" ) ;
else
printf ( "The two pointers do not point to the same location\n" ) ;
return 0 ;
}
```

The two pointers point to the same location

Two-Dimensional Arrays

- The two-dimensional array is also called a matrix.

```c
# include <stdio.h>
int main( )
{
int stud[ 4 ][ 2 ] ;
int i, j ;
for ( i = 0 ; i <= 3 ; i++ )
{
printf ( "Enter roll no. and marks" ) ;
scanf ( "%d %d", &stud[ i ][ 0 ], &stud[ i ][ 1 ] ) ;
}
for ( i = 0 ; i <= 3 ; i++ )
printf ( "%d %d\n", stud[ i ][ 0 ], stud[ i ][ 1 ] ) ;
return 0 ;
}
```

Conceptual View

	column no. 0	column no. 1
row no. 0	1234	56
row no. 1	1212	33
row no. 2	1434	80
row no. 3	1312	78

Initializing a Two-Dimensional Array

```
int stud[ 4 ][ 2 ] = {
    { 1234, 56 },
    { 1212, 33 },
    { 1434, 80 },
    { 1312, 78 } } ;
```

Or

```
int stud[ 4 ][ 2 ] = { 1234, 56, 1212, 33, 1434, 80,
1312, 78 } ;
```

Initializing a Two-Dimensional Array

- While initializing a 2-D array, it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.

- The following declaration is acceptable

```
int arr[ 2 ][ 3 ] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][ 3 ] = { 12, 34, 23, 45, 56, 45 } ;
```

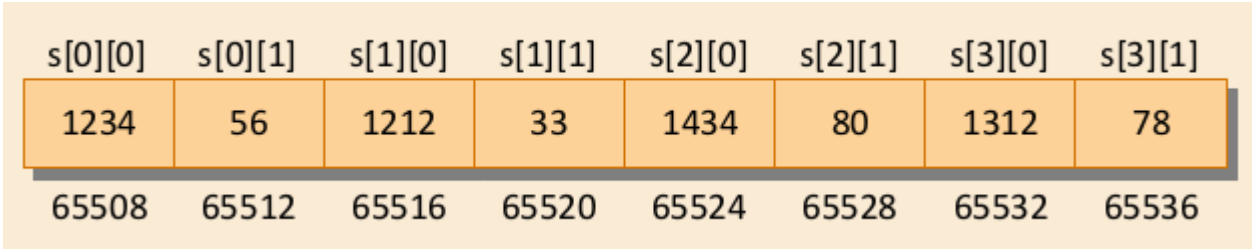
- The following declaration is not acceptable

```
int arr[ 2 ][ ] = { 12, 34, 23, 45, 56, 45 } ;
int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;
```

Memory Map of a 2D Array

- The arrangement of array elements of a 2D array in memory

  int s[4][2];



- int s[5][2] ;
- can be considered as setting up an array of 5 elements, each of which is a one-dimensional array containing 2 integers.

Pointers and Two-Dimensional Arrays

```c
# include <stdio.h>
int main( )
{
int s[ 4 ][ 2 ] = {
{ 1234, 56 },
{ 1212, 33 },
{ 1434, 80 },
{ 1312, 78 }
} ;
int i ;
for ( i = 0 ; i <= 3 ; i++ )
printf("Address of %d th 1-D array = %u\n", i, s[ i ] ) ;
return 0 ;
}
```

```
Address of 0 th 1-D array = 1676564960
Address of 1 th 1-D array = 1676564968
Address of 2 th 1-D array = 1676564976
Address of 3 th 1-D array = 1676564984


------------------
(program exited with code: 0)
Press return to continue
```

Pointers and Two-Dimensional Arrays

```c
# include <stdio.h>
int main( )
{
    int s[ 4 ][ 2 ] = {
{ 1234, 56 },
{ 1212, 33 },
{ 1434, 80 },
{ 1312, 78 }
} ;
int i, j ;
for ( i = 0 ; i <= 3 ; i++ )
{
for ( j = 0 ; j <= 1 ; j++ )
printf ( "%d ", *( *( s + i ) + j ) ) ;
printf ( "\n" ) ;
}
return 0 ;
}
```

```
1234 56
1212 33
1434 80
1312 78


-----------------
(program exited with code: 0)
Press return to continue
```

Exercise

- C Program to print lower diagonal of a matrix
- C program to interchange the rows in the matrix
- C program to find the sum of main and opposite diagonal elements of a matrix
- C program to print the upper triangular matrix
- C Program to Transpose a Matrix