2-D Arrays

Matrices for mathematical calculations.

Physical representation for science.

Example 2darray

Example 2darray illustrates the initialization and use of a C-style 2-D array.

Declare and Initialize

```
#define ROWS 3
#define COLS 5
typedef int Table[ROWS][COLS];

Table A ={ 13, 22,  9, 23, 12,
           17,  5, 24, 31, 55,
            4, 19, 29, 41, 61};
```


typedef

- You can create your own type with typedef
- Makes a shortcut you can use as a type
- Once made you can use as a type declaration (make a variable, type a parameter etc.)
- improves readability
- easier to change (one place to change!)

Row major order

- C stores multidimensional arrays in row major order. By that we mean that

Table myTable[3][5]

- has three arrays, each of 5 values (three rows of 5 values)

in fact, one block of memory

myTable[3][3]={{1,2,3},{4,5},{6,7,8}};

Indexing: myTable[i][j] is located at:

```
*((myTable[i] + j))
(*(myTable + i)[j])
*((*(myTable + i)) + j)
*(&myTable[0][0] + 3*i + j)
```


display()

For each **row "i"**

For each **column "j"** (in each row)

output the element **A [i][j]**

display()

```
void display(Table X){
  for (int i=0; i < ROWS; i++){
    for (int j=0; j < COLS; j++ )
      printf(" X[%d][%d]: %d",i,j,X[i][j]);
    printf("\n");
  }
  printf("\n");
}
```

display()

i=0

j= 0, 1, 2, 3, 4

output: 13 22 9 23 12 // first row

---

i=1

j= 0, 1, 2, 3, 4

output: 17 5 24 31 55 // second row

display()

```
void display( Table X ){
    for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++)
          printf(" X[%d][%d]:
%d",i,j,X[i][j]);
       printf('\n');
       }
     printf('\n');
}
```


main()

```
Declare A
Display(A)
Change A      // i + j
Display(A)
Change A      // *10
Display(A)
```


Change A: i + j

For each **row "i"**

For each **column "j"** (in each row)

A [i][j] = i + j ;

Change A: i + j

```
for (int i=0; i<ROWS; i++)
 {                          for (int j=0;
 j<COLS; j++)                    A[i][j] = i + j;

 }
```


A[i][j] = i + j;

i=0

j= 0, 1, 2, 3, 4

i + j : 0 1 2 3 4 // first row

i=1

j= 0, 1, 2, 3, 4

i + j: 1 2 3 4 5 // second row

A[i][j] = i + j;

```
0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
```


Change A: *10

For each **row "i"**

For each **column "j"** (in each row)

BACKWARDS: right to left

A [i][j] = A[i][**j -1**] * 10;

Change A: *10

```
for (int i=0; i<ROWS; i++)
   {                              for (int
   j=COLS-1; j>0; j--)
         A[i][j] = A[i][j-1] * 10;
   }
```


A[i][j] = A[i][j-1]*10;

```
 i=0
 j=            4, 3, 2, 1  // no Zero
A[i][j-1] : 3, 2, 1, 0
 A[i][j]  : 30,20,10, 0
```

No Zero because of the `[j-1]` index.

A[i][j] = A[i][j-1]*10;

```
0  1  2  3  4
1  2  3  4  5
2  3  4  5  6
                    0  0 10 20
30
1 10 20 30 40
2 20 30 40 50
```


passing 2d arrays

- In passing an array, only the first array size is “unbound”. The second (and any subsequent) must be specified!
- Thus you can pass:

```
int myFun(int list[][10]);
int myFun(int (*list)[10]);
```

- first is greatly preferred!