Pointers and 2-D Arrays

Initialization of 2-D Array

```
#include <stdio.h>
#define MAXROW 5
#define MAXCOL 5
int main()
{
  int a[MAXROW][MAXCOL] = {{0,1,2,3,4},
                           {10,11,12,13,14},
                           {20,21,22,23,24},
                           {30,31,32,33,34},
                           {40,41,42,43,44}},
  int b[MAXROW][MAXCOL] = {{10,20,30},  {40,50,60,70,80}},
  int c[MAXROW][MAXCOL} = {0,1,2,3,4,10,11,12,13,14,20,21,22,
                           23,24,30,31,32,33,34,40,41,42,43,44},
  int d[][MAXCOL] = {2,4,6,8,0,4,6,8,0,2};
  :
  :
```

What does array name mean in 2-D array?

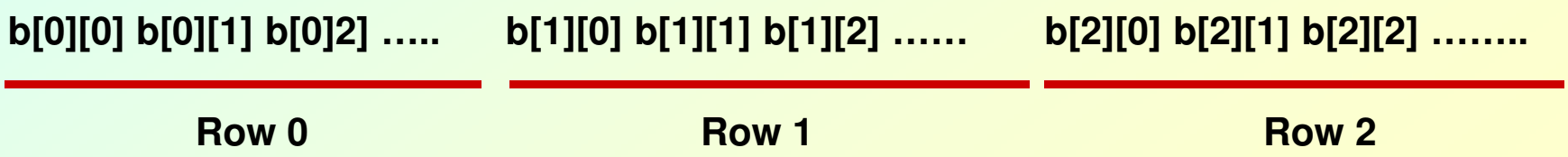
```
int  a[10], b[5][3];
```

- We know that '`a`' is a constant pointer whose value is the address of the 0th element of the array `a[10]`.
- Similarly, `a+i` is the address of the ith element of the array.
- What is the meaning of '`b`' and what is its arithmetic?

How is a 2-D array is stored in memory?

- 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

Element `b[i][j]` :: allocated memory location at address `x+(i*c+j)*k`



Arithmetic of 'b'

- `b` is the address of the 0^{th} row.
- `b+1` is the address of the 1^{st} row.
- `b+2` is the address of the 2^{nd} row.
- In general, `b+i` will represent the starting address of the i^{th} row.
- The size of a row will be:

  c × sizeof(int) = 5 × 4 = 20 bytes

  where c is the number of columns.

An example program

```
#include <stdio.h>
int main()
{
   int a[10], b[3][5];
   printf ("a: %u \tb: %u\n", a, b);
   printf ("a+1: %u \tb+1: %u\n", a+1, b+1);
   printf ("a+2: %u \tb+2: %u\n", a+2, b+2);
   printf ("a+3: %u \tb+3: %u\n", a+3, b+3);
}
```

Output

```
a: 3217738332    b: 3217738272
a+1: 3217738336  b+1: 3217738292
a+2: 3217738340  b+2: 3217738312
a+3: 3217738344  b+3: 3217738332
```

Type of ‘b’

- ‘`b`’ is a pointer constant of type `int[][5]`, that is, a row of five integers.
- If such a pointer is incremented by one, it goes up by `5×sizeof(int)` bytes.

Arithmetic of `*(b+i)`

- If '`b`' is the address of the 0th row, `*b` is the 0th row itself.
  - A row may be viewed as a 1-D array, so `*b` is the address of the 0th element of the 0th row.
- Similarly, `b+i` is the address of the ith row, `*(b+i)` is the ith row.
  - So `*(b+i)` is the address of the 0th element of the ith row.

- If `*b` is the address of the 0th element of the 0th row, `*b+1` is the address of the 1st element of the 0th row.
- Similarly, `*b+j` is the address of the jth element of the 0th row.
- The difference between `b+1` and `b` is 20 bytes, but the difference between `*b+1` and `*b` is the `sizeof(int)`, that is, 4 bytes.

- So, `*(b+i)` is the address of the 0th element of the i th row.
- Thus, `*(b+i)+j` is the address of the j th element of the i th row.
  - That is, same as `&b[i][j]`.

```
*(b+i)+j is equivalent to &b[i][j]
```

Some Equivalences

```
*(*(b + i) + j)   ⟹   b[i][j]

*(b + i) + j      ⟹   &b[i][j]

*(b[i] + j)       ⟹   b[i][j]

b[i] + j          ⟹   &b[i][j]

(*(b + i))[j]     ⟹   b[i][j]
```

Calculation of the address of `b[i][j]`

```
int  b[3][5]
```

- The C compiler can calculate the address of the j^{th} element of the i^{th} row using the following formula:

  `b + k (5i + j)`

  where `k = sizeof(int)`.

- The compiler needs the following:
  - Value of row and column indices
  - The number of columns
  - The size of the data type.

1-D Array and Formal Parameter

- Consider the declaration: `int a[10];`
  - The array name ‘a’ is a constant pointer.
  - The formal parameter: `int x[]` or `int *x` is a pointer variable of the corresponding type, where the address of an array location is copied into the function.
  - These two information are sufficient for the compiler to calculate the address of `x[i]`.

Formal parameter for 2-D Array

- **Consider the declaration:** `int b[ROW][COL];`
  - The C compiler needs the following information to calculate the address of `b[i][j]` (values of `i` and `j` are information local to the function):
    - Starting address '`b`'
    - The data type of the array elements, that is, '`int`'
    - The number of columns '`COL`'

An example

```
#include <stdio.h>

void transpose (int x[][3],
                int n)
{
   int  p, q, t;

   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;
       }
}
```

```
main()
{
  int a[3][3], p, q;

  for (p=0; p<3; p++)
    for (q=0; q<3; q++)
      scanf ("%d", &a[p][q]);
  transpose (a, 3);
  for (p=0; p<3; p++)
  {
    printf ("\n");
    for (q=0; q<3; q++)
      printf ("%d  ", a[p][q]);
  }
}
```