Lecture 3:
Representation of Multidimensional Arrays
What is an Array?

- An array is a fixed size sequential collection of elements of identical types.
- A multidimensional array is treated as an array of arrays.
  - Let a be a k-dimensional array; the elements of a can be accessed using the following syntax:
    
    \[
    a_{i_1, i_2, \ldots, i_k}
    \]

The following loop stores 0 into each location in two dimensional array A :

```c
int row, column;
int A[3][4];
for (row = 0; row < 3; row++)
{
    for (column = 0; column < 4; column++)
    {
        A[row][column] = 0;
    }
}
```
Definition of a Multidimensional Array

- **One-dimensional** arrays are linear containers.

Multi-dimensional Arrays
Two-Dimensional Array

If you create an array \( A = \text{new int}[3][4] \), you should think of it as a “matrix” with 3 rows and 4 columns.

But in reality, \( A \) holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.
Storage Allocation

The storage arrangement shown in this example uses the array subscript, also called the array indices.

Array declaration: `int a[3][4];`

Array elements:

```
  a[0][0]  a[0][1]  a[0][2]  a[0][3]
  a[1][0]  a[1][1]  a[1][2]  a[1][3]
```
Array size

• In a matrix which is defined as
  \[a_{0}[upper_1][upper_2]...[upper_{n-1}],\]
  the number of items is:

  \[\prod_{i=0}^{n-1} upper^i\]

**Example:** What is the number of items in \(a[20][20][1]\)?
Memory Storage

- There are two types of placement for multidimensional arrays in memory:
  - Row major ordering
  - Column major ordering

**Example:** In an array which is defined as $A[upper_0][upper_1]$, if the memory address of $A[0][0]$ is $\alpha$, then what is the memory address of $A[i][0]$ (according to row major ordering)?
Memory Storage

- There are two types of placement for multidimensional arrays in memory:
  - Row major ordering
  - Column major ordering

Example: In an array which is defined as $A[0][1]$, if the memory address of $A[0][0]$ is $\alpha$, then what is the memory address of $A[i][0]$ (according to row major ordering)?

$$\alpha + i \times \text{upper}_1$$
Memory Storage

• For a three-dimensional array $A[upper_{0}][upper_{1}][upper_{2}]$
what is the memory storage like?

• **Example:** char $y[2][2][4]$

  which slice? which row? which column?

• What is the memory address of $y[1][1][3]$ if the memory address of $y[0][0][0]$ α?
Memory Storage

The memory address of \( a[i][0][0] \) is:

\[
\alpha + i \times upper_1 \times upper_2
\]

if the memory address of \( a[0][0][0] \) is \( \alpha \). Therefore, the memory address of \( a[i][j][k] \) becomes:

\[
\alpha + i \times upper_1 \times upper_2 + j \times upper_2 + k
\]

The memory address of \( a[i_0][i_1][i_2]...[i_{n-1}] \) is:

\[
\alpha + \sum_{j=0}^{n-1} i_j a_j
\]

\[
a_j = \prod_{k=j+1}^{n-1} upper_k \quad 0 \leq j \leq n-1
\]

\[
a_{n-1} = 1
\]
Pointers and Multi-dimensional Arrays
int A[5]

int *p = A;

Print p  // 200
Print *p  // 2
Print *(p+2)  // 6

200 204 208 212 216

```c
int A[5]

int *p = A;
Print A  // 200  
Print *A  // 2  
Print *(A+2)  // 6  

200 204 208 212 216
2 4 6 8 10


*(A+i) is same as A[i]  
(A+i) is same as &A[i]  
```
```c
int A[5]

int *p = A;

Print A  // 200
Print *A  // 2
Print *(A+2)  // 6
```

```
P = A;  \checkmark
A = P;  \times
```

```
2  4  6  8  10
```

\( *(A+i) \text{ is same as } A[i] \)

\( (A+i) \text{ is same as } A[A[i]] \)
int A[5]
A[0] \rightarrow \text{int}
A[1]

int B[2][3]
B[0] \rightarrow 1-D \text{ arrays of 3 integers}
B[1]
int B[2][3]

B[0] → 1-D arrays of 3 integers

B[1]

int *p = B;  // will return a pointer to 1-D array of 3 integers

400  412

B[0]  B[1]
int B[2][3]

B[0] \rightarrow 1-D arrays of 3 integers

B[1] \rightarrow 

int *p = B;  

will return a pointer to 1-D array of 3 integers

int (*p)[3] = B;  

\checkmark
```c
int B[2][3]
```

B[0] \rightarrow 1-D arrays of 3 integers

B[1] \rightarrow 1-D arrays of 3 integers

```c
int (*p)[3] = B;
```

\[
\downarrow \\
\text{will return a pointer to 1-D array of 3 integers}
\]

Print B or &B[0] \# 400

Print *B or B[0] or &B[0][0] \# 400
```plaintext
int B[2][3]

B[0]  \rightarrow  1-D arrays of 3 integers
B[1]

int (*p)[3] = B;

Print B or &B[0]  // 400
Print *B or B[0] or &B[0][0]  // 400
Print B+1  // 400+12 = 412

& &B[17]
```

```c
int B[2][3]

B[0] \rightarrow 1-D arrays of 3 integers
B[1]

int (*p)[3] = B;

Print B or &B[0] // 400
Print *B or B[0] or &B[0][0] // 400
Print B+1 or &B[1] // 412
Print *(B+1) or B[1] or &B[1][0] // 412
```
```c
int B[2][3]

B[0] \rightarrow 1-D arrays of 3 integers
B[1]

int (*p)[3] = B;

Print B or &B[0] // 400
Print *B or B[0] or &B[0][0] // 400
Print B+1 or &B[1] // 412
Print *(B+1) or B[1] or &B[1][0] // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2] // 420
```
int B[2][3]
B[0] \rightarrow 1-D arrays of 3 integers
B[1]

int (*P)[3] = B;

Print B or &B[0] \quad // 400
Print *B or B[0] or &B[0][0] \quad // 400
Print B+1 or &B[1] \quad // 412
Print *(B+1) or B[1] or &B[1][0] \quad // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2] \quad // 420
Print *(P+1)
int B[2][3]
B[0] \rightarrow 1-D arrays
B[1] of 3 integers
int (*p)[3] = B;
Print B or &B[0] // 400
Print *B or B[0] or &B[0][0] // 400
Print B+1 or &B[1] // 412
Print *(B+1) or B[1] or &B[1][0] // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2] // 420
Print *(B+1) B \rightarrow int (*)[3]
B[0] \rightarrow int *
&B[0][1]
int B[2][3]

B[0] \rightarrow 1-D arrays of 3 integers

int (*p)[3] = B;

Print B or &B[0] // 400
Print *B or B[0] or &B[0][0] // 400
Print B+1 or &B[1] // 412
Print *(B+1) or B[1] or &B[1][0] // 412
Print *(B+1)+2 or B[1]+2 or &B[1][2] // 420
Print *(B+1) // 3

B[0][1]
int B[2][3]

For 2-D array

\[ B[i][j] = * (B[i] + j) \]
\[ = * (\cdot (B + i) + j) \]
Pointers and multi-dimensional arrays

```c
int B[2][3]
int (*P)[3] = B;  // declaring pointer to 1-D array of 3 integers
int *P = B;      // X
```
Pointers and multi-dimensional arrays

```c
int B[2][3]
int (*P)[3] = B;  // Pointer to row
Print B // 400  // Print B
Print *B // 400  // Print the value of the pointer
Print B[0] // 400  // Print B[0]
```
Pointers and multi-dimensional arrays

```c
int B[2][3]
int (*P)[3] = B;
Print &B //400
Print *B //400
Print B[0] //400
Print &B[0][0] //400
```
Pointers and multi-dimensional arrays

```c
int C[3][2][2]
```

```
2 5 7 9 3 4 6 1 0 8 11 13
```
Pointers and multi-dimensional arrays

```c
int C[3][2][2]

int *(p)[2][2] = C;

Print int *(p)[2][2] // 800

Print *C or C[0] or &C[0][0] // 800

int *(p)[2]
```
Pointers and multi-dimensional arrays

int c[3][2][2]

int (*p)[2][2] = c;

Print c // 800

Print *c or c[0] or &c[0][0]

\[
c[i][j][k] = * (c[i][j] + k) = *(c[i] + i) + k
\]

= *(c[0][0] + c[0][1] + c[0][2])
Pointers and multi-dimensional arrays

```c
int C[3][2][2]
int (*P)[2][2] = C;

Print C   // 800
Print *(C[0][2] + 1) or C[0][1][1] // 9
```
Pointers and multi-dimensional arrays

```c
int C[3][2][2]
int (*p)[2][2] = C;

Print C  // 800
Print int (*)[2][2]  // 800
Print +C or C[0] or &C[0][0]  // 800
Print * (C[0][1] + 1) or C[0][1][1]  // 9
Print * (C[1] + 1) or C[1][1] or &C[1][1][0]  // 824
```

2 5 7 9 3 4 6 1 0 8 11 13
// Pointers and multi-dimensional arrays
#include<stdio.h>
int main()
{
    int C[3][2][2]=
        {{2,5},{7,9}},
        {{3,4},{6,1}},
        {{0,8},{11,13}};
    printf("%d %d %d %d", C, *C, C[0], &C[0][0]);
}
References

• BBM 201 Notes by Mustafa Ege
• Lecture Videos: www.mycodeschool.com/videos/pointers-and-arrays