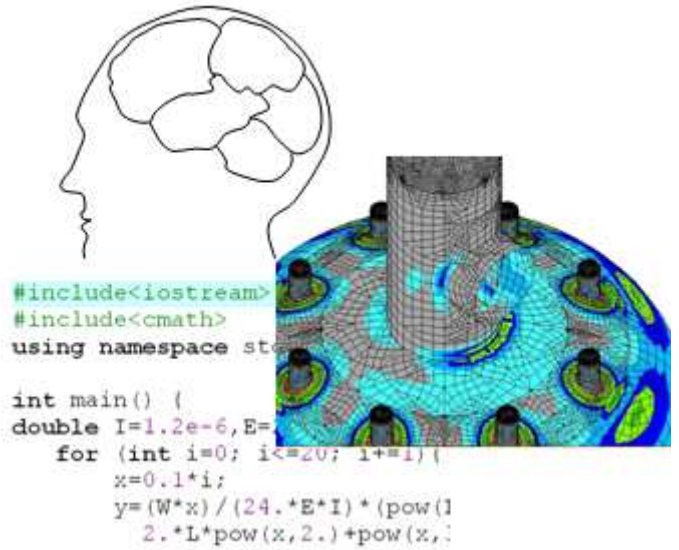
ME 240 Computation for Mechanical Engineering



Arrays

Multi-Dimensional

Content

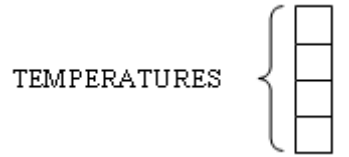
This week we will study Multi Dimensional Arrays:

- Concept
- Declaration, Initialization, Assignment
- Matrix Operations Using Two Dimensional Arrays

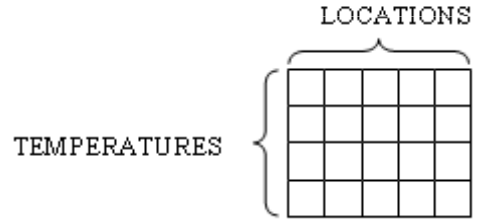
Concept and Declarations of Multi Dimensional Arrays

An array of arrays is called a *multidimensional array*.

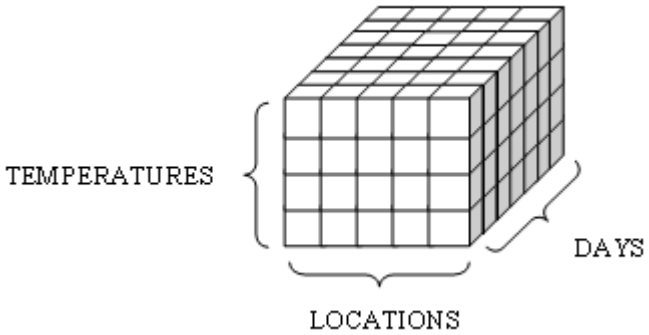
E.g. a one-dimensional array of one-dimensional arrays is a two dimensional array.




```
// 4-element one-dimensional array
int t[4];
```




```
// 20-element two-dimensional array
int t[4][5];
```




```
// 140-element three-dimensional array
int t[4][5][7];
```

Initilization of Multi Dimensional Arrays

To initialize a two-dimensional array, it is easiest to use nested braces, with each set of numbers representing a row:

```
int b[3][5] =
   {{ 1, 2, 3, 4, 5},   //first row
    { 6, 7, 8, 9,10},   //second row
    {11,12,13,14,15}};  //third row
```

When the C++ compiler processes this list, it actually ignores the inner braces altogether. Therefore, `b` array can be also defined as;

```
int b[3][5] =
   { 1, 2, 3, 4, 5,   //first row
     6, 7, 8, 9,10,   //second row
    11,12,13,14,15};  //third row
```

However, we highly recommend you use them anyway for readability purposes.

Two-dimensional arrays with initializer lists can omit (only) the first size specification:

```
int b[ ][5] =
   {{ 1, 2, 3, 4, 5},
    { 6, 7, 8, 9,10},
    {11,12,13,14,15}};
```

However, the following is not allowed:

```
int b[ ][ ] =
   {{ 1, 2, 3, 4, 5},
    { 6, 7, 8, 9,10},
    {11,12,13,14,15}};
```

Because the inner parenthesis are ignored, the compiler can not tell whether you intend to declare a 1×15, 3×5, 15×1, or 5×3 array in this case:

Just like normal arrays, multidimensional arrays can still be initialized to **0** as follows:

```
int b[3][5] = { 0 };
```

Note that this only works if you explicitly declare the size of the array!

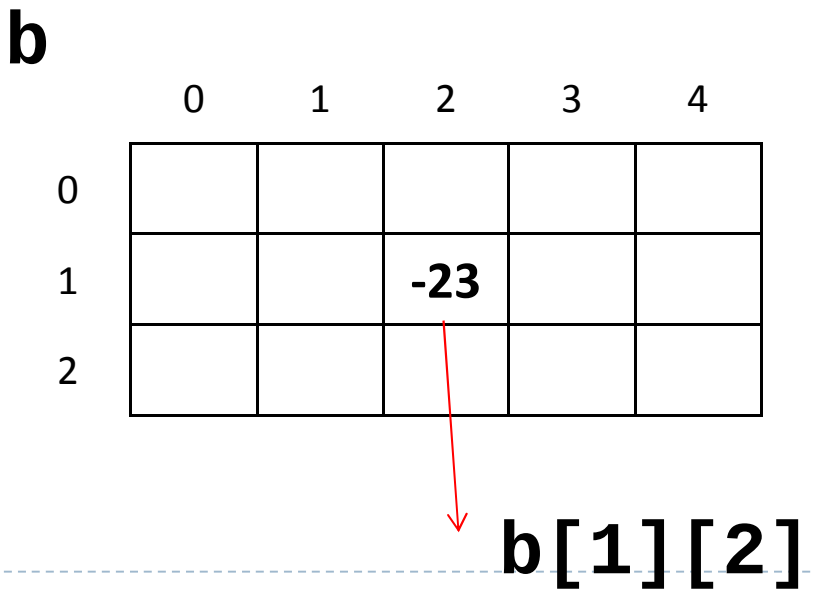
Otherwise, you will get a two-dimensional array with 1 row.

Assignments of Multi Dimensional Arrays

The statement

```
b[1][2] = -23;
```

assigns the value 23.4 to the element identified by (1,2), and can be visualised as in the Figure below.



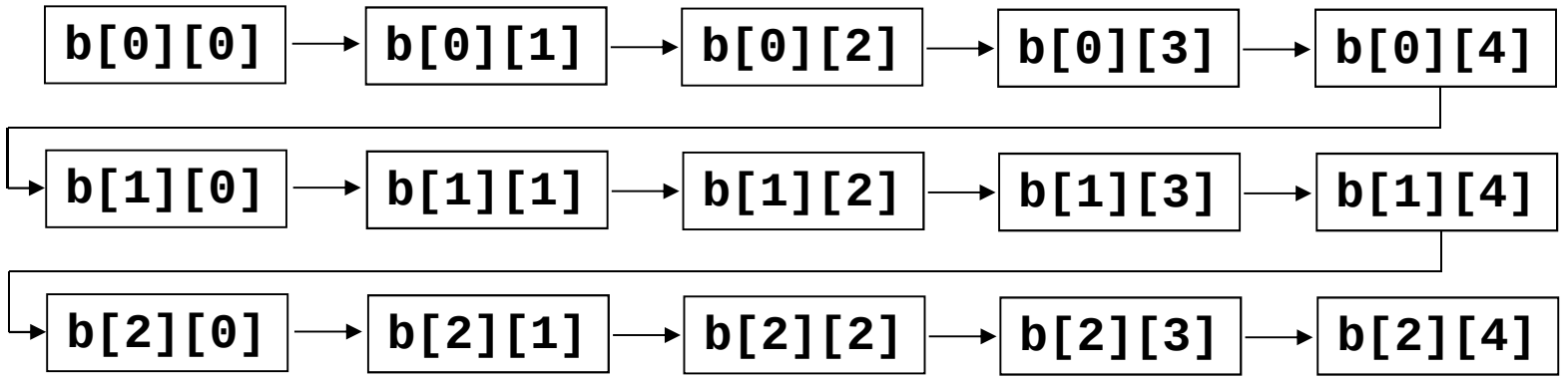
Inputs and Outputs of Multi Dimensional Arrays

Row-wise Input

Accessing all of the elements of a two-dimensional array requires two loops: one for the row, and one for the column.

In the row-wise process, the outer loop is used for the row index;

```
for (int i = 0; i < nNumRows; i++){
   for (int j = 0; j < nNumCols; j++){
       cout<<"input the element,"<<i<<","<<j<<endl;
       cin >> b[i][j];}}
```

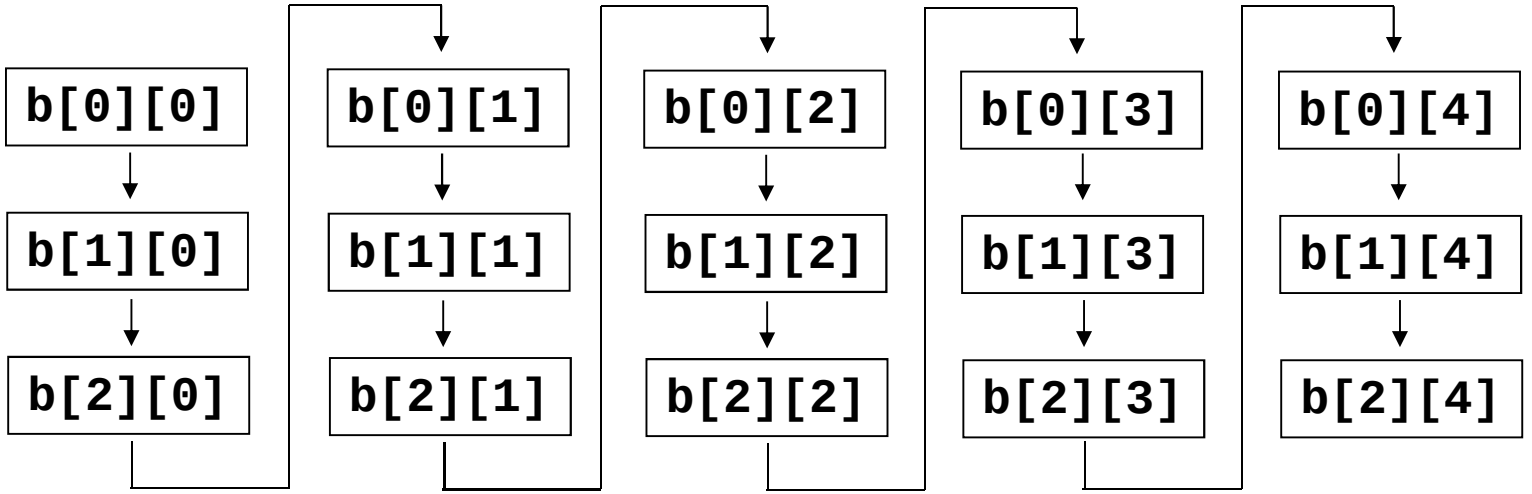


Inputs and Outputs of Multi Dimensional Arrays

Column-wise Input

In the column-wise process, the outer loop is used for the column index;

```
for (int j = 0; j < nNumCols; j++){
   for (int i = 0; i < nNumRows; i++){
       cout<<"input the element,"<<i<<","<<j<<endl;
       cin >> b[i][j];} }
```



The program shows row-wise input and output

```
#include <iostream>
using namespace std;

int main(){
  const int nNumRows=2,nNumCols=3;
  int b[nNumRows][nNumCols];

for (int i = 0; i < nNumRows; i++){
   for (int j = 0; j < nNumCols; j++){
       cout<<"input the element,"<<i<<","<<j<<endl;
       cin >> b[i][j];}}

for (int i = 0; i < nNumRows; i++) {
  for (int j = 0; j < nNumCols; j++){
         cout << b[i][j]<<"\t";};
         cout<<endl;}
system("pause"); }
```

The program shows column-wise input and row wise-output

```
#include <iostream>
using namespace std;

int main(){
  const int nNumRows=2,nNumCols=3;
  int b[nNumRows][nNumCols];

for (int j = 0; j < nNumCols; j++){
   for (int i = 0; i < nNumRows; i++){
       cout<<"input the element,"<<i<<","<<j<<endl;
       cin >> b[i][j];} }

for (int i = 0; i < nNumRows; i++) {
  for (int j = 0; j < nNumCols; j++){
         cout << b[i][j]<<"\t";};
         cout<<endl;}
system("pause"); }
```

Matrix Operations using Multi Dimensional Arrays

Matrix Transpose;

```
#include <iostream>
using namespace std;

int main(){
int m[2][3]={2,3,1,4,2,7},mt[3][2];

cout<<"transpose of m= \n";
 for(int i=0;i<3;i++){
   for(int j=0;j<2;j++){
    mt[i][j]=m[j][i];
    cout<<mt[i][j]<<"\t";
   }
    cout<<"\n";
 }
system("pause"); }
```

Matrix Operations using Multi Dimensional Arrays

Working with Diagonal Elements;

A program that displays the following matrices

7	0	0	0		0	0	0	7
0	7	0	0		0	0	7	0
0	0	7	0		0	7	0	0
0	0	0	7		7	0	0	0

```
#include <iostream>
using namespace std;
int main(){
    int m[4][4]={0},n[4][4]={0};
    for(int i=0;i<4;i++){
       m[i][i]=7; }
    cout<<"the first matrix    "<<endl;
    cout<<"====================="<<endl;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
         cout<<m[i][j]<<"\t";
      } cout<<"\n"; }
    for(int i=0;i<=3;i++){
       n[i][3-i]=7; }
    cout<<"the second matrix    "<<endl;
    cout<<"======================"<<endl;
    for(int i=0;i<4;i++){
      for(int j=0;j<4;j++){
         cout<<n[i][j]<<"\t";
      } cout<<"\n"; }
system("pause");    }
```

Matrix Operations using Multi Dimensional Arrays

Matrix Addition and Subtraction;

```
#include <iostream>
using namespace std;
int main(){
double m[2][3]={2.3,3.4,1.2,4.7,2.6,7.8},
       n[2][3]={1.1,2.4,6.7,3.7,1.4,3.6};
cout<<"m+n= \n";
 for(int i=0;i<2;i++){
   for(int j=0;j<3;j++){
    cout<<m[i][j]+n[i][j]<<"\t";
   }
    cout<<"\n"; }
   cout<<"m-n= \n";
    for(int i=0;i<2;i++){
      for(int j=0;j<3;j++){
       cout<<m[i][j]-n[i][j]<<"\t";
      }
       cout<<"\n"; }
   system("pause");    }
```