Instructor created

6.28 Functions with 2D array parameters

There are several ways to pass a 2D array to a function.

Code Editor 6.28.1: Specify the size of columns of 2D array

Click run.

Load default template...

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

void printArray(int a[][2], int row, int col) {  // must specify the column size in the 2d array parameter.
   cout << "[";
   for (int i=0; i<row; i++)
   {
      cout << "[" << a[i][0];
      for (int j=1; j<col; j++) {
         cout << ", " << a[i][j];
      }

      cout << "]";
      if (i<row-1) {
         cout << "," << endl;
      }
   }
```

Run

Code Editor 6.28.2: Pass array containing pointers

Click run.

Load default template...

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

void printArray(int *a[], int row, int col) {  // must specify the column size in the 2d array parameter.
   cout << "[";
   for (int i=0; i<row; i++)
   {
      cout << "[" << a[i][0];
      for (int j=1; j<col; j++) {
         cout << ", " << a[i][j];
      }

      cout << "]";
      if (i<row-1) {
         cout << "," << endl;
      }
   }
```

Run

If the array is dynamically allocated, then you may use single pointer to typecast the 2D array:

Code Editor 6.28.3: Using Single pointer to typecast the 2D array

Click run.

Load default template...

```
#include <iostream>

using namespace std;


void print2D(int *arr, int m, int n)          //function prototype
{
   for (int i=0; i<m; i++)
   {
      for (int j=0; j<n; j++)
      {
         cout<<*((arr+i*n) + j)<<" ";    //printing each element using pointer arithmetic

      }
      cout<<endl;
   }
}
```

Run

Notes: 1-D array is *(Array+i) is same as Array[i]

For 2-D array, Array[row][col] is same as *(Array+row_i*column_size)+column_j) using pointer arithmetic

Code Editor 6.28.4: Using Double pointers (pointer to pointer)

Click run.

Load default template...

```
#include <iostream>

using namespace std;

void func(int **arr, int row, int col)        //function prototype
 {
    for (int i=0; i<row; i++)
    {
       for(int j=0 ; j<col; j++)
       {
          cout<<arr[i][j]<<" ";   //printing each element of array
       }
       cout<<endl;
    }
 }
int main()
 {
   int row, colm;
```

```
2 2 1 1 2 2
```

Run

Arrays of pointers:

Pointers to pointers have a few uses. The most common use is to dynamically allocate an array of pointers:

int **array = new int*[10];

This allocates an array of 10 int pointers which are rows in the above code. It works just like a standard dynamically allocated array, except the array elements are of type "pointer to integer" instead of an integer.

Figure 6.28.1: Arrays of pointers

```
for(int count=0; count<10; count++)
   array[count] = new int[5]; //these are our columns
```

We can then access our array as usual,

array[6][4] = 5;