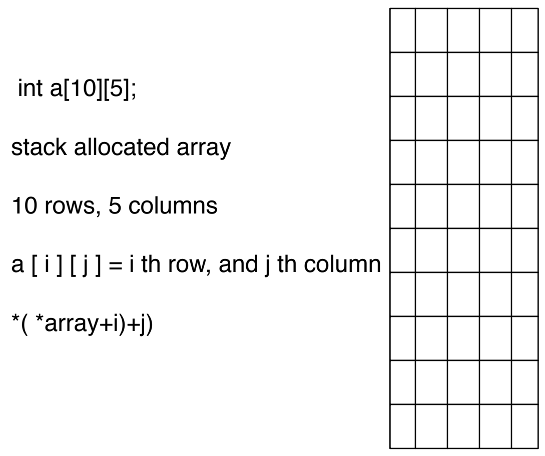
MultiDimensional Arrays

Stack Allocated 2D Array of int

```
int a[10][5];
```

This creates a stack allocated array with 10 rows and 5 columns. An individual element of the array can be accessed by specifying both a row number and a column number.

```
a [ i ] [ j ] refers to the i-th row and j-th column.
```



The memory associated with the array will be reclaimed when the array goes out of scope.

Stack Allocated Array of int *

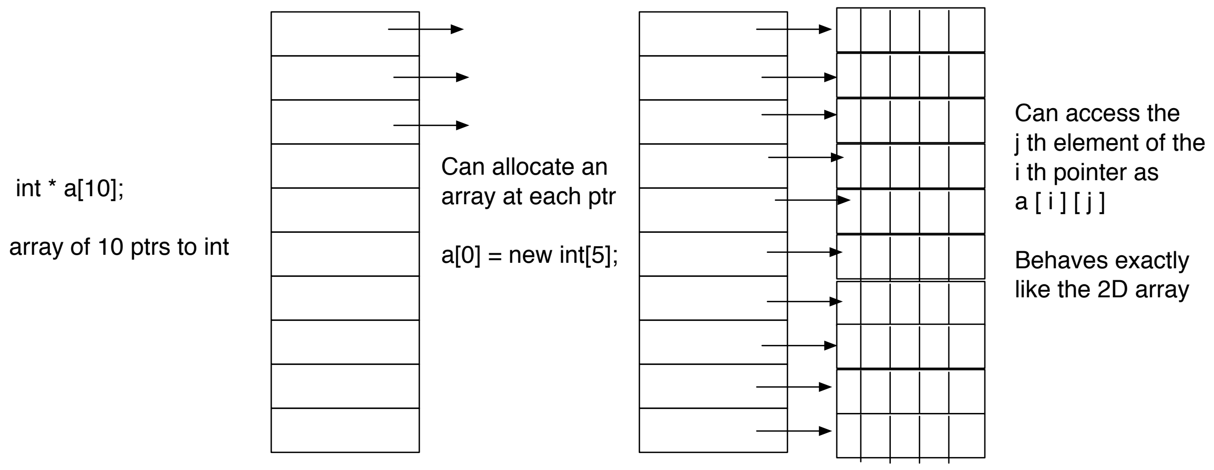
```
int * a[10];
```

This creates a stack allocated array of 10 elements where each element is an int*. Each element can be allocated as follows:

```
for (int i=0; i < 10; i ++) {
   a[i] = new int[5];
}
```

This means that each element of this stack allocated array is a dynamically allocated array of ints.

Contrast this with the previous array (int a[10][5]) which was entirely stack allocated. The two arrays hold exactly the same number of int values which can be accessed identically (a [i] [j] accesses the int at the i-th row and j-th column).



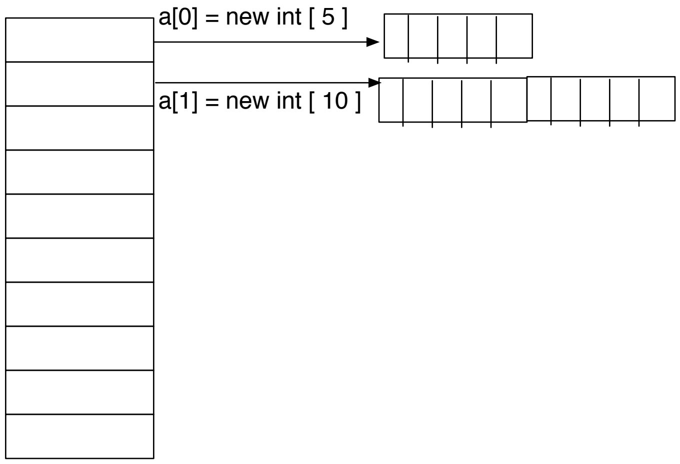
The memory associated with each element of this array will NOT be automatically reclaimed, since the elements have been dynamically allocated. We must delete these dynamically allocated arrays ourselves:

```
for (int i=0; i < 10; i ++) {
   delete [ ] a[i];
}
```

Note: we do not have to delete the array itself since it is stack allocated and the memory associated with the array will be reclaimed when it goes out of scope.

```
Nothing is restricting us to have each ‘‘row" array be of the
    same size
```

Since each element of the array is a dynamically allocated array, we can create these arrays with different dimensions



Dynamically allocated array of int *

We can allocated the original array in the heap. Each element of this array is an int * which can be dynamically allocated to hold an array of int.

```
int **a;
a = new int*[10]; //dynamically create an array of 10 int *
for ( int i = 0 ; i < 10; i++){ //for each row
    a [i] = new int [5]; //allocate 5 cols
    for (int j = 0 ; j < 5; j ++) { //for each column in a[i]
        cin >> a [i] [j];
    }
}
```

The array above holds exactly the same number of elements as the previous arrays. However, all arrays are dynamically allocated (in the heap) which means that we must reclaim the memory consumed ourselves

```
for (int i = 0; i < 10; i ++) {
   delete [] a[i];
}
delete [] a;
```