CS 162
Intro to Computer Science II

Lecture 6

Dynamic array (cont.)

Struct

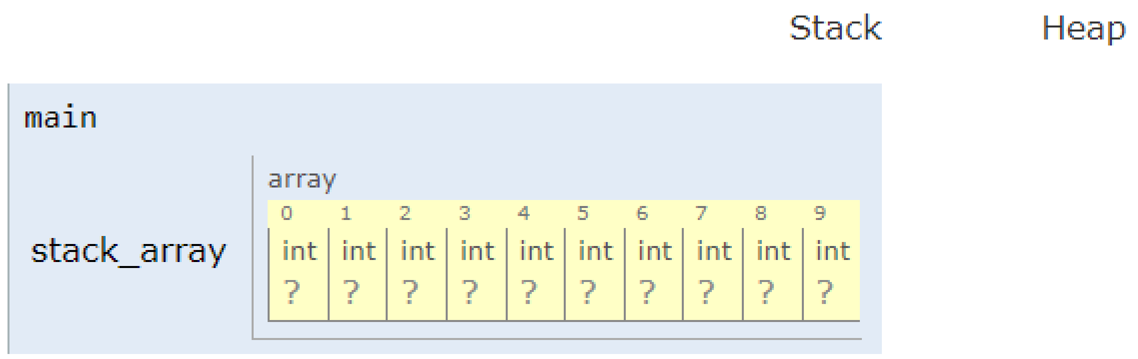
1/26/24

Odds and Ends

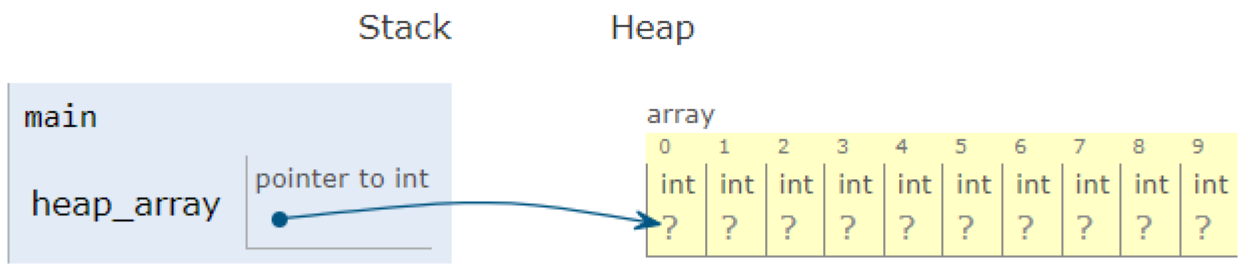
- Assignment 1 due Sunday midnight via TEACH
- Sign up for demos! (use your OSU email)

Static vs. Dynamic 1-D arrays...

```
int main() {
  int stack_array[10];

  return 0;
}
```



```
int main() {
  int *heap_array = new int [10];

  return 0;
}
```

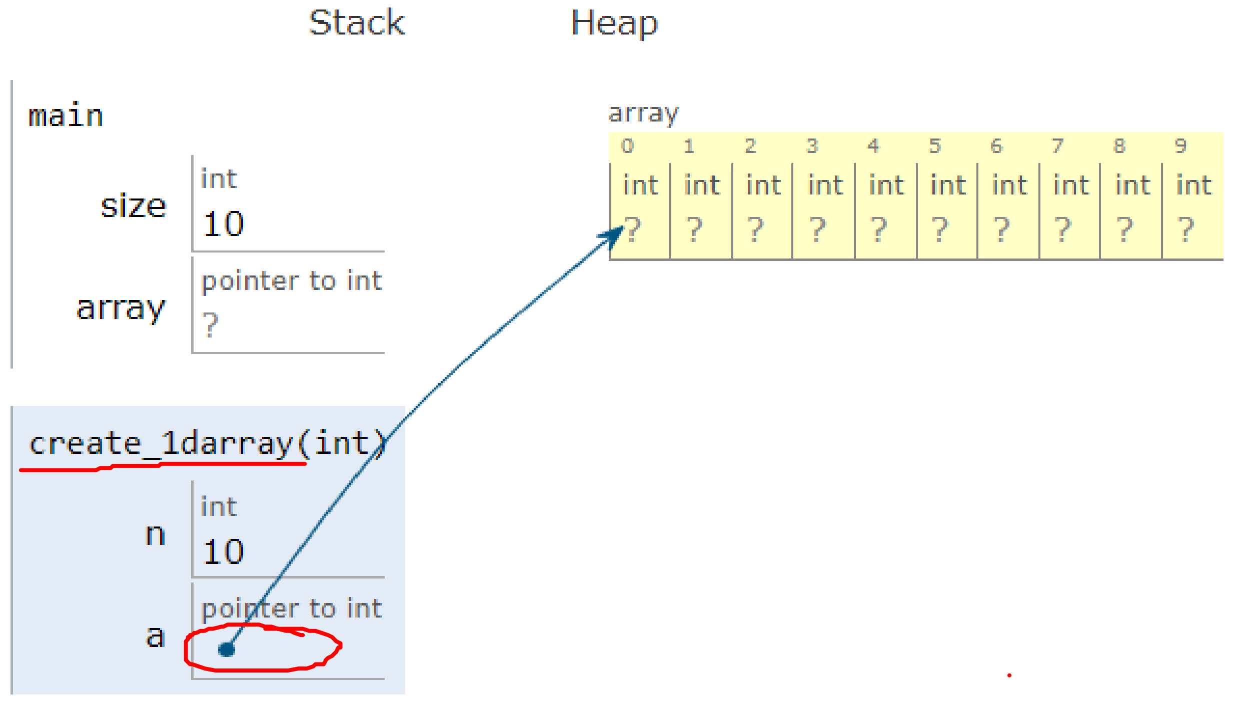


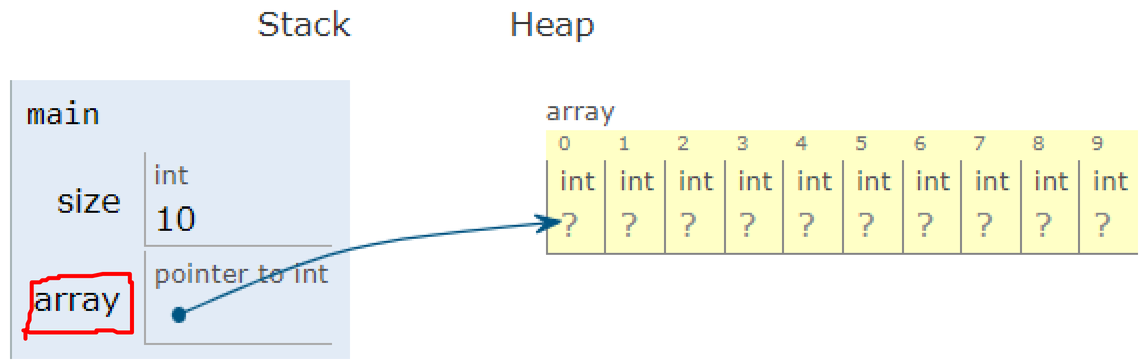
Exercise

- How do I initialize an int array in a function?
- How can I print the contents of the int array in a function?
- How would I create a dynamic int array using a function? (3 ways)
  - int* create_array1(int size);
  - void create_array2(int *&array, int size);
  - void create_array3(int ** array, int size);

Create 1-D Array in Functions

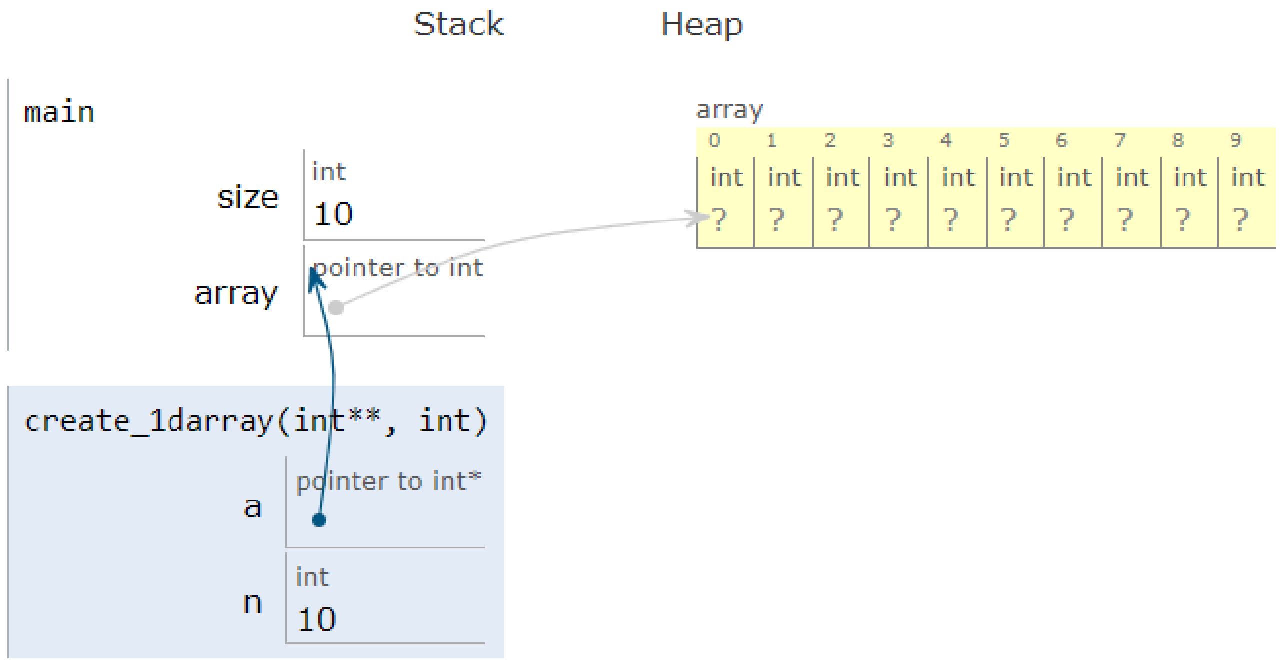
```
int main() {
    int *array;
    …
    array = create_1darray(size);
    …
}
int *create_1darray(int n) {
     int *a = new int [n];
     return a;
   }
```

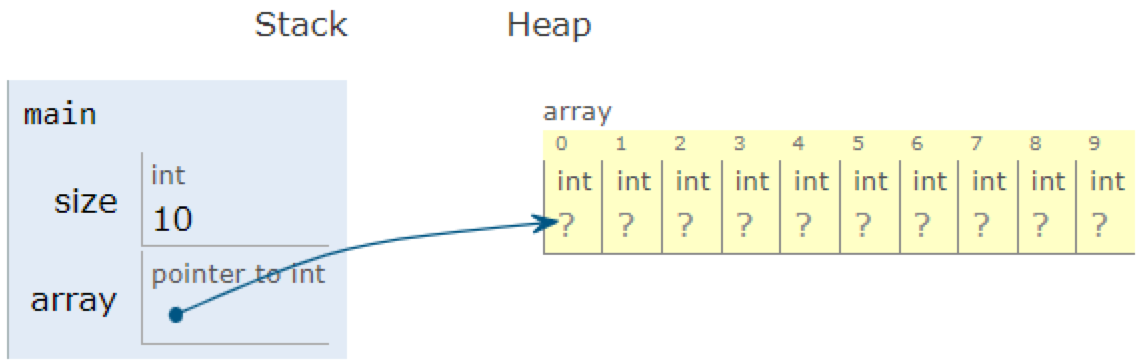




Create 1-D Array in Functions

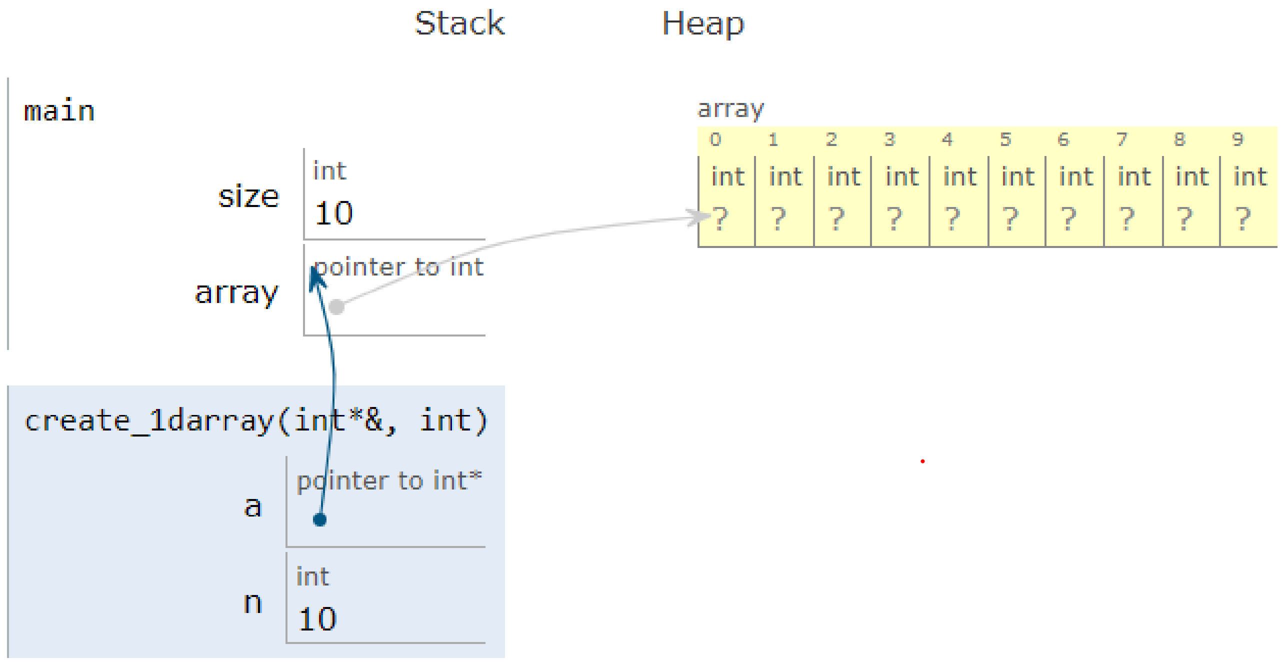
```
int main() {
   int *array;
   …
create_1darray(&array, size);
   …
}
void create_1darray(int **a, int n) {
    *a = new int [n];
}
```

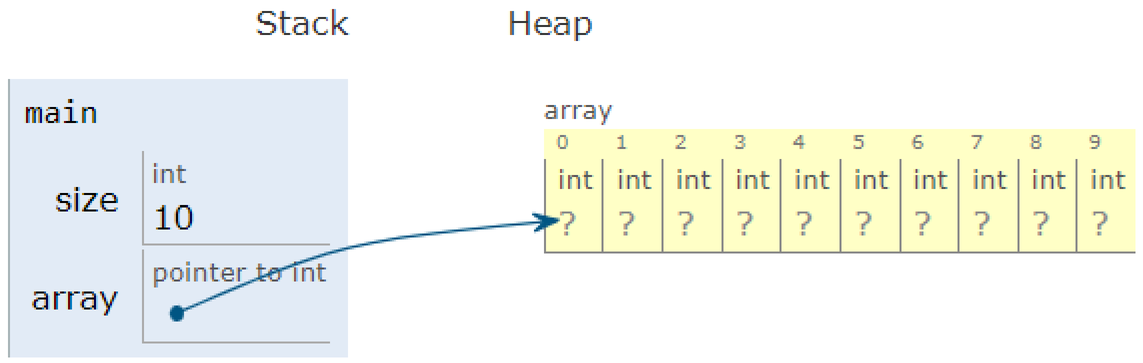




Create 1-D Array in Functions

```
int main() {
   int *array;
   …
create_1darray(array, size);
   …
}
void create_1darray(int *&a, int n) {
   a = new int [n];

}
```





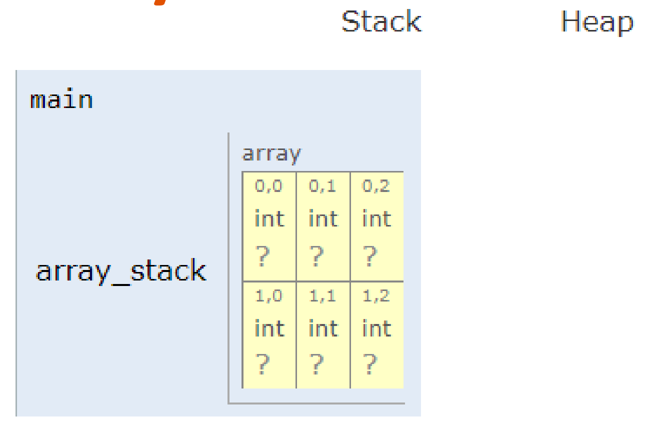
Lecture Topics:

- 2D dynamic array

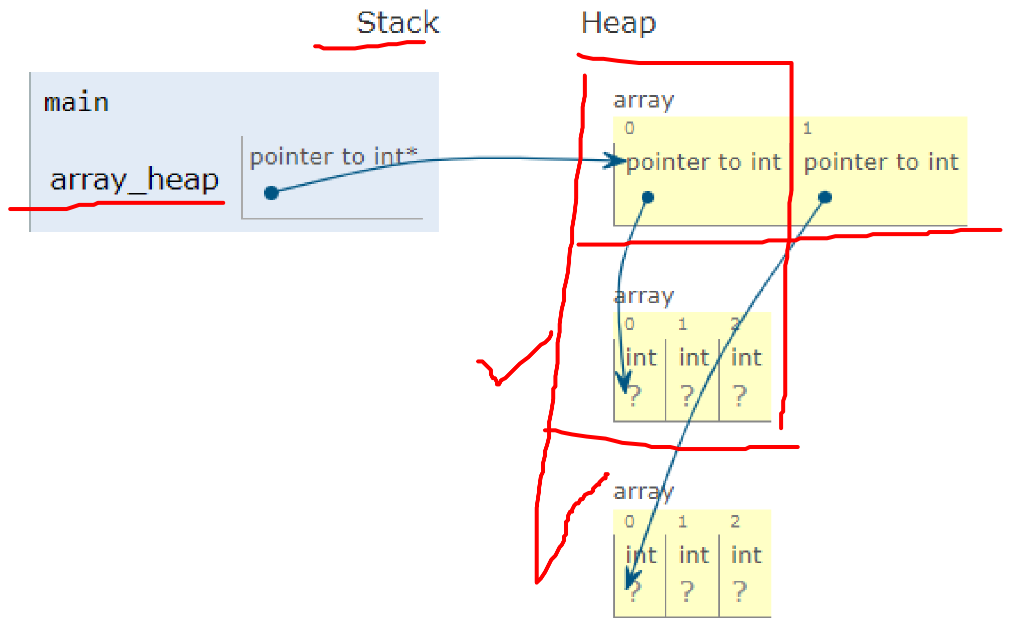
- Structs

Static vs. Dynamic 2-D arrays...

```
int main() {
  int array_stack[2][3];

  return 0;
}
```

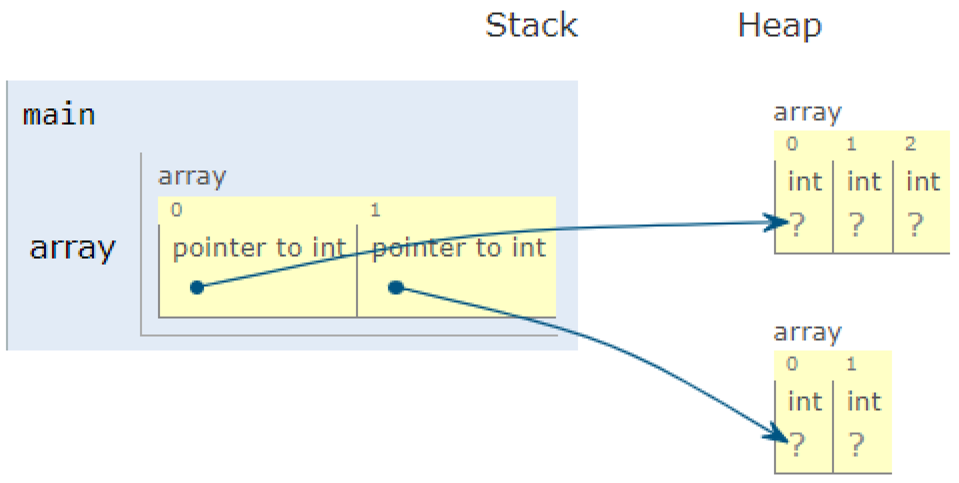


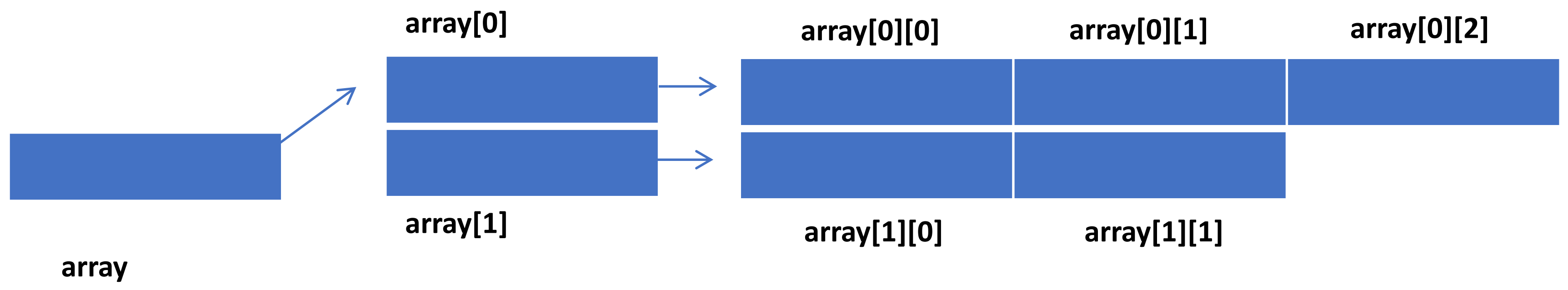
```
int main() {
  int **array_heap = new int* [2];
  for(int i = 0; i < 2; i++)
    array_heap[i] = new int [3];

  return 0;
}
```



Jagged Arrays

```
int *array[2];
array[0] = new int[3];
array[1] = new int[2];
```





Passing a 2-D Array (Static)

```
int main() {
    int array[5][5];
    …
    pass_2darray(array);
    OR
    pass_2darray(array, 5);
    …
}
void pass_2darray(int a[5][5]) {
    cout << “Array at zero: ” << a[0][0] << endl;
}
OR
void pass_2darray(int a[][5], int row) {
    cout << “Array at zero: ” << a[0][0] << endl;
}
```

Passing a 2-D Array (Dynamic)

```
int main() {
    int **array;
    …
    pass_2darray(array, row, col);
    …
}
void pass_2darray(int *a[], int row, int col) {
     cout << “Array at zero: ” << a[0][0] << endl;
}
```

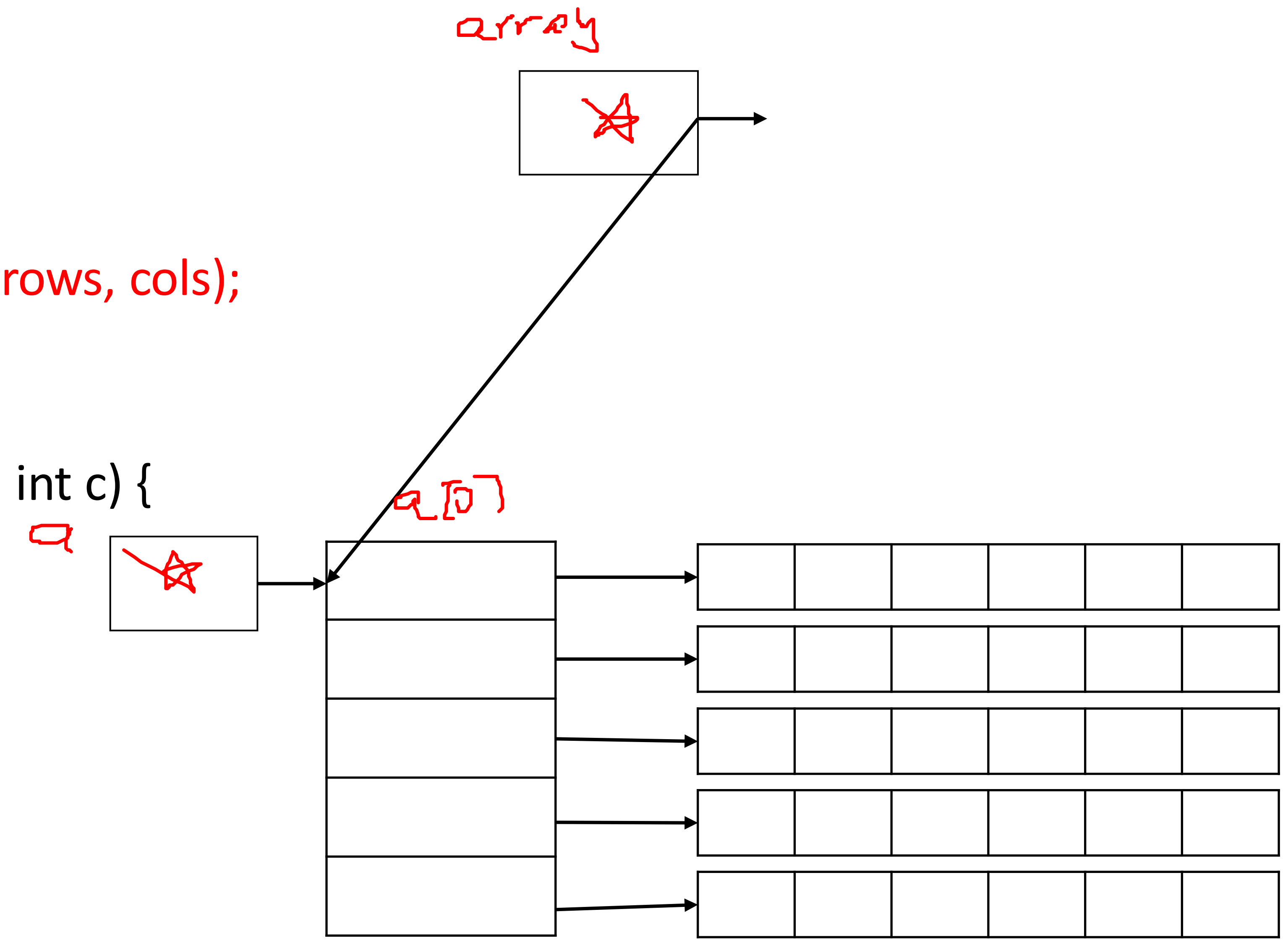
OR

✓

```
void pass_2darray(int **a, int row, int col) {
    cout << “Array at zero: ” << a[0][0] << endl;
}
```

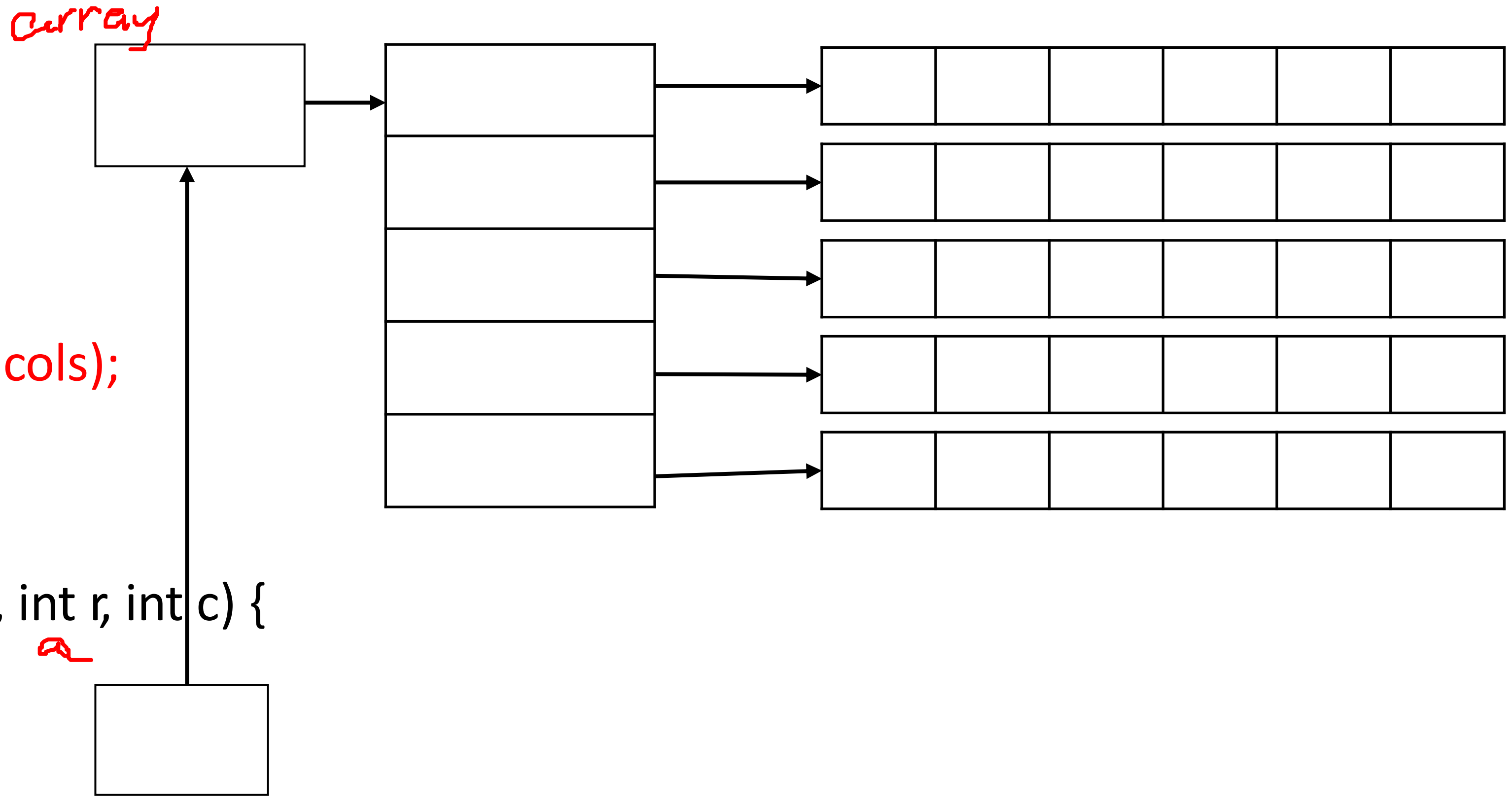
Create 2-D Array in Functions

```
int main() {
    int **array;
    …
    array = create_2darray(rows, cols);
    …
}
int **create_2darray(int r, int c) {
     int **a;
     a = new int*[r];
     for(int i=0; i<r; i++)
        a[i] = new int[c];
     return a;
   }
```



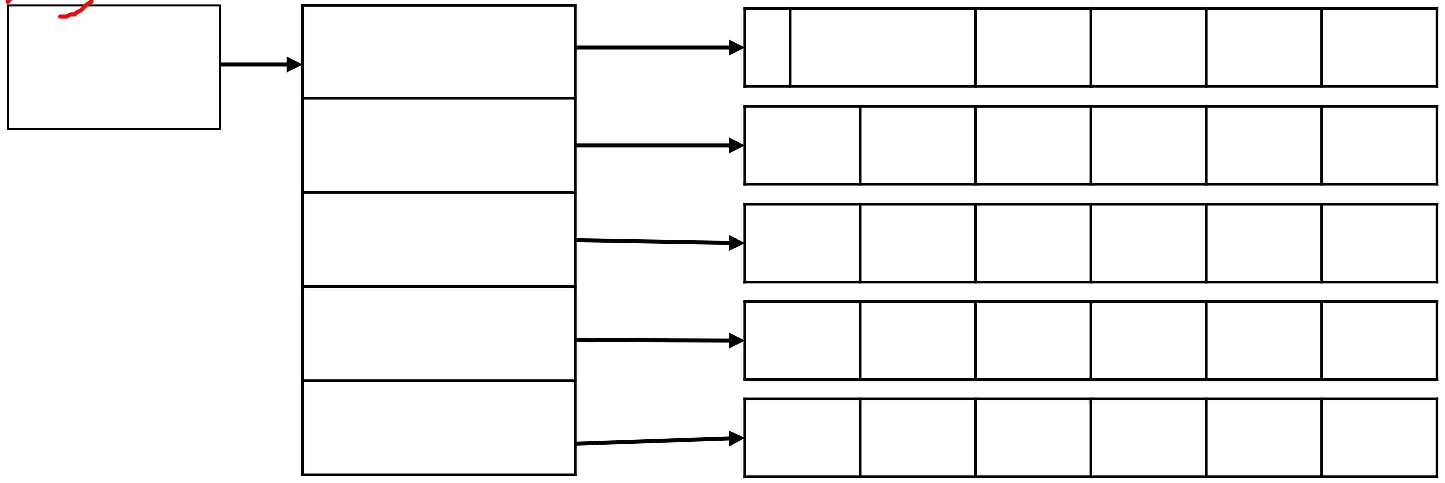
Create 2-D Array in Functions

```
int main() {
  int **array;
  …
create_2darray(&array, rows, cols);
  …
}
void create_2darray(int ***a, int r, int c) {
   *a = new int*[r];
   for(int i=0; i<r; i++)
      (*a)[i] = new int[c];
}
```



Create 2-D Array in Functions

a, array



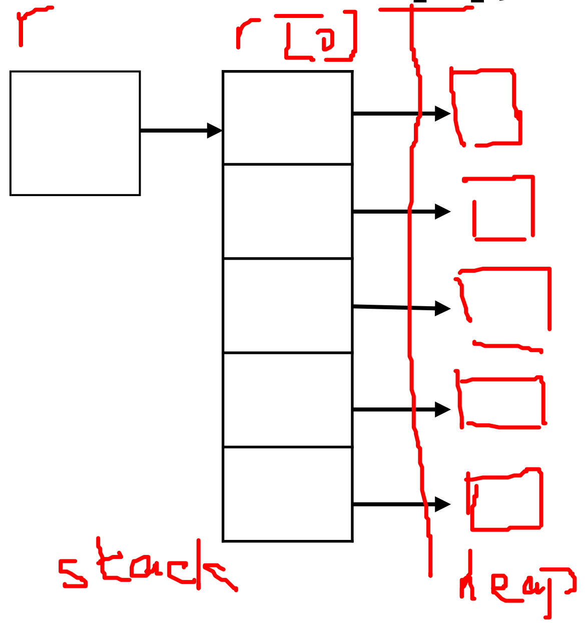
```
int main() {
   int **array;
   …
create_2darray(array, rows, cols);
   …
}
void create_2darray(int **&a, int r, int c) {
    a = new int*[r];
    for(int i=0; i<r; i++)
        a[i] = new int[c];
}
```

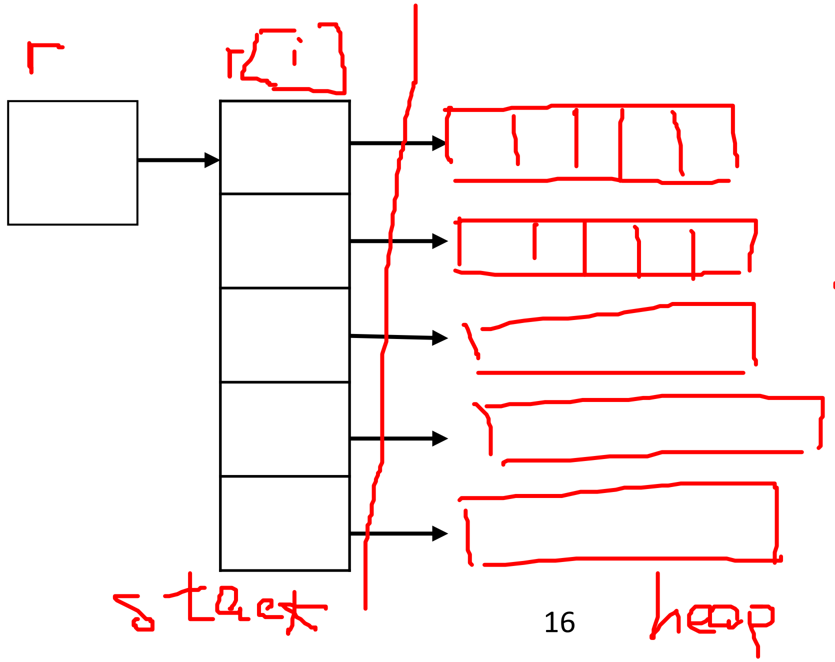
How does freeing memory work in 2D arrays?

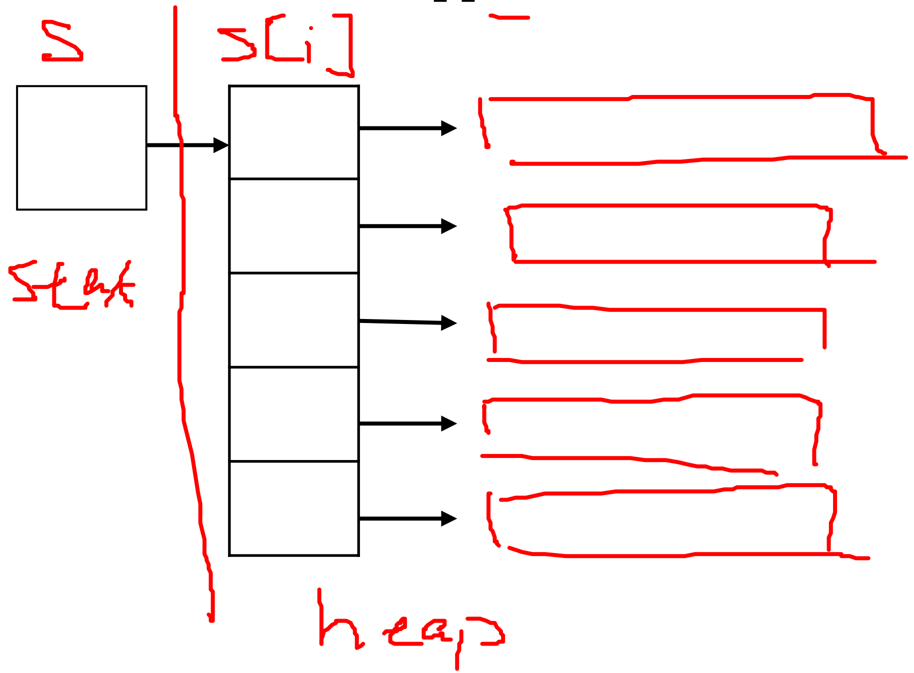
```
int *r[5], **s;

for(int i=0; i < 5; i++)
    r[i]=new int;
for(int i=0; i < 5; i++)
    delete r[i];
```

```
for(int i=0; i < 5; i++)
    r[i]=new int[5];
for(int i=0; i < 5; i++)
    delete [] r[i];
```

```
s=new int*[5];
for(int i=0; i < 5;
i++)
    s[i]=new int[5];
for(int i=0; i < 5;
i++)
    delete [] s[i];
delete [] s;
```





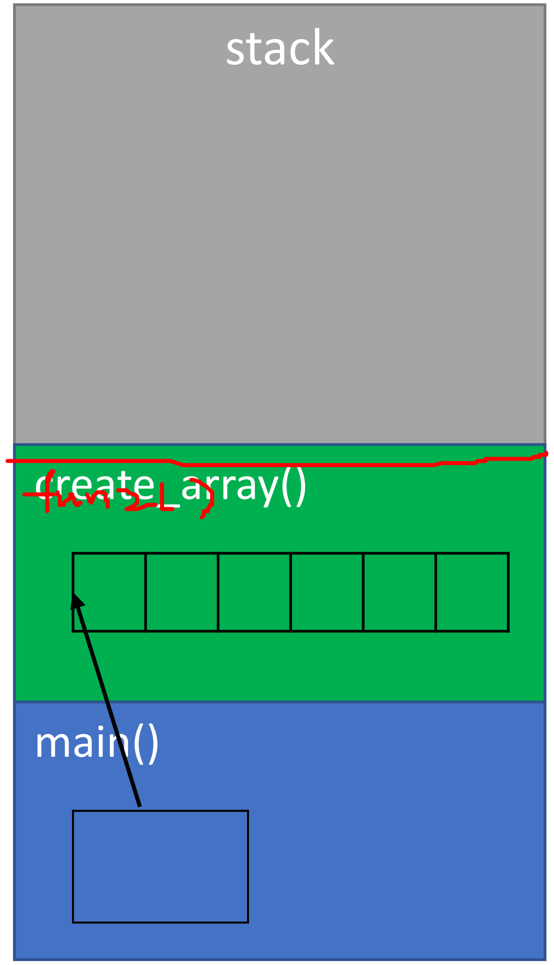


Recap: Array

- Dynamic Arrays: creates on the heap and their size may change during runtime
  - 1-dimensional dynamic array: `int *heap_array = new int [10];`
    - To free (delete):
      ```
      delete [] heap_array;
      heap_array = NULL;
      ```
  - Creating 2-dimensional dynamic array: (This allocates a 4x5 2D array)
    ```
    int row = 4, col = 5;
    int **matrix = new int* [row];
    for (int i = 0; i < row; i++){
            matrix[i] = new int[col];
    }
    ```
  - To free (delete) a 2D array, reverse the work you did when you allocated it:
    ```
    for (int i = 0; i < row; i++){
            delete [] matrix[i];
            matrix[i] = NULL;
    }
    delete [] matrix;
    matrix = NULL;
    ```

Why can’t you create a static 1D/2D array and return its address?

```
int main() {
    int *array;
    …
    array = create_array(size);
     fun2( )
    …
}
int *create_array(int size) {
     int a[size];
     return a;
  }
```



Lecture Topics:

- Structs

Structures

- Data Structures so far…
  - Variables
  - Arrays

- What if we want mixed types?
  - Record: name, age, weight, etc. of a person
  - Use **struct** type

Structs

- User defined composite data type
- Container which holds many variables of different types
- Can be used as any other data type with some extra features
- The instances created by such data type are called objects (items)

How to define a struct?

```
// definition of a Book struct
struct Book {
        int pages;
        string title; // a string inside the struct
        int num_authors;
        string* authors; // a pointer to a string
};
```

member var.

data type

```
// declare a Book object (item)
Book text_book;

// declare and initialize at the same time
Book b1 = {.pages = 150, .title = "Harry Potter", .num_authors = 2};
//or
Book b1 = {150, "Harry Potter", 2};
```

Note: in order, non-skip

Working with structs

- Can use the same way as any other type
- The dot operator(.) allows us to access the member variables

```
Book bookshelf[10];
for (int i = 0; i < 10; ++i){
     bookshelf[i].num_pages = 100;
     bookshelf[i].title = “Harry Potter”;
     bookshelf[i].num_authors = 2;
     bookshelf[i].authors = new string[2];
}
```

Using pointers with structs

```
Book bk1; //statically allocated
Book* bk_ptr = &bk1;

//dereference the pointer and access the data member
(*bk_ptr).title = "Harry Potter";

//a shortcut to dereference the pointer to the struct
// the arrow (->) operator
bk_ptr -> title = "The Cars";
bk_ptr -> num_pages = 259;

//this works for objects on the heap as well
Book* bk_ptr2 = new Book;
bk_ptr2 -> title = "Transformers";
```

Demo