# Lecture 37: More C++

## Today

- Reading
  - Bailey Chapter 16 (Graphs)
- Objectives
  - Lessons from lab
  - Recap primitive arrays
  - Iterators
  - (Graph algorithms)

## + Lessons from Lab

- *Great job to Hong and Yubai for tracking down all 78 calls to the copy constructor*!
  - The first 42 happen in the for-loop
  - The remaining 36 happen in the while loop
  - Calling `rest()` invokes the copy constructor. Why?!
- It is possible to call `delete` on a pointer and yet still be able to access the data the pointer points to!
- The `delete` function first calls the destructor of the memory being freed
- Care is needed when converting from objects to pointers

## + Copy Constructor

- Constructs a new object from an existing object
- Examples of when the copy constructor is called:
  - `IntCell copy = original;`
  - `IntCell copy(original);`
  - `a formal parameter of a call-by-value function`
  - `an object returned by value`
- It would not be called in this instance:

```
IntCell copy;

copy = original;
```

Remember this slide from Lecture 33?!

## + Recap: Primitive arrays in C++

- To declare an array:

  int arr[3]; // notice where the brackets go

- Compiler computes how many bytes needed for array
- The name of the array is a constant pointer to the first element of the array
- Correction
  - Correct syntax: arr + i
  - Incorrect syntax: arr+4*i

## + Recap: Implications

- Cannot assign to an array

```
int array1[3];
int array2[3];
array2 = array1;
```

- Saying array2 == array1 tests memory equality
- An array (i.e. the pointer to the first element) is passed by value.
  - The overall effect, however, is that the array itself is passed by reference
  - Changes to the formal parameters show up in the input arguments

## + Dynamically allocated arrays

- Use the `new[ ]` operator!
  - Just like the new operator but for arrays
  - creates an array of objects on the heap
  - There is a corresponding `delete[ ]` operator

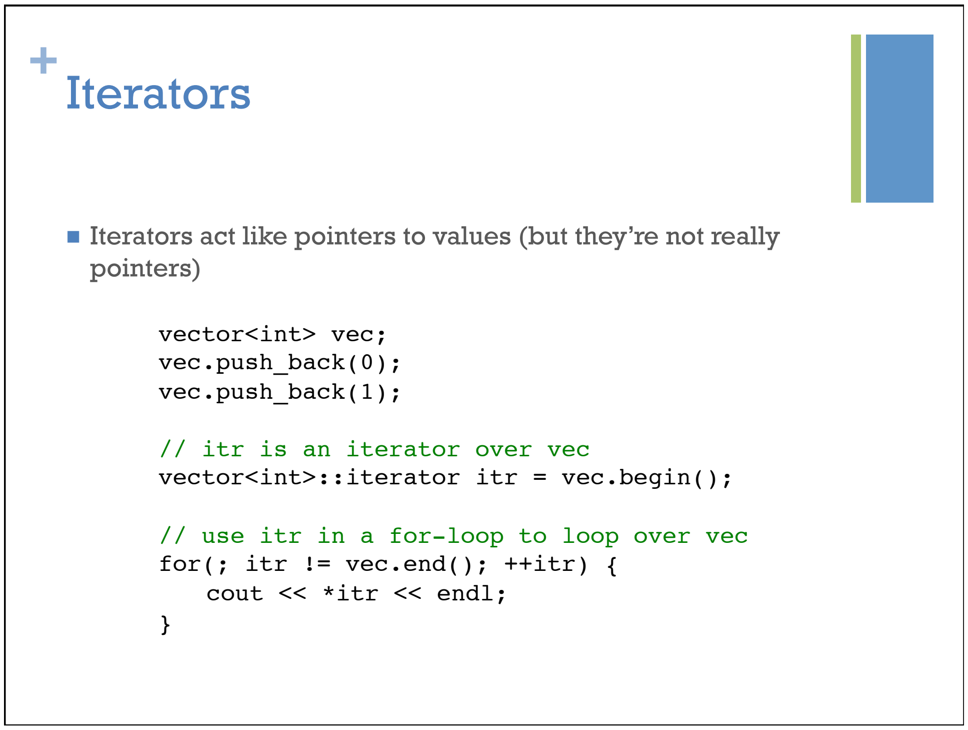
```
void my_function() {
    int SIZE = 3;
    int array1[SIZE];             // allocated on the stack
    int *array2 = new int[SIZE];  // allocated on the heap
}
```

After my_function returns, what memory is freed and what is not?

## + Iterators

- In Java, an iterator is a class that allows you to iterate over the elements of a collection
  - Implements the Iterator interface
  - What are the methods specified in the Iterator interface?
- In C++, every class has its own iterator type

```
vector<int>::iterator   // iterator for a vector
vector<int>::const_iterator // can't modify vector
map<int,int>::iterator // iterator for a map
```



## Iterators

- Iterators act like pointers to values (but they're not really pointers)

```
vector<int> vec;
vec.push_back(0);
vec.push_back(1);

// itr is an iterator over vec
vector<int>::iterator itr = vec.begin();

// use itr in a for-loop to loop over vec
for(; itr != vec.end(); ++itr) {
    cout << *itr << endl;
}
```