Effective C++

Return to Vector

MEC++ 1. Distinguish between pointers and references

- No such thing as a Null reference -- reference must refer to a valid object.

```
Vector &v;      // Error!
Vector *v;      // Valid but dangerous
```

- If an object must exist make a reference; if it might be Null, make a pointer.

EC++ 21. Use `const` whenever possible

What’s pointed to is constant	Pointer is constant
char *	p = “Hello”;
const char *	p = “Hello”;
char *	**const** p = “Hello”;
const char *	**const** p = “Hello”;

Function declarations:

- Return value: `const int foo()...`
- Parameters: `int foo (const char *...)`
- Function itself: `int foo () const {….}`

EC++ 3. Use `new` and `delete` instead of `malloc` and `free`

- What is dynamic memory?
- What does `new` do?
- What does `delete` do?
- Who cares?

EC++ 4. Use the same form in corresponding call to `new` and `delete`

- How are these different?

```
Item * myList = new Item;
Item * myList = new Item[newSize];
```

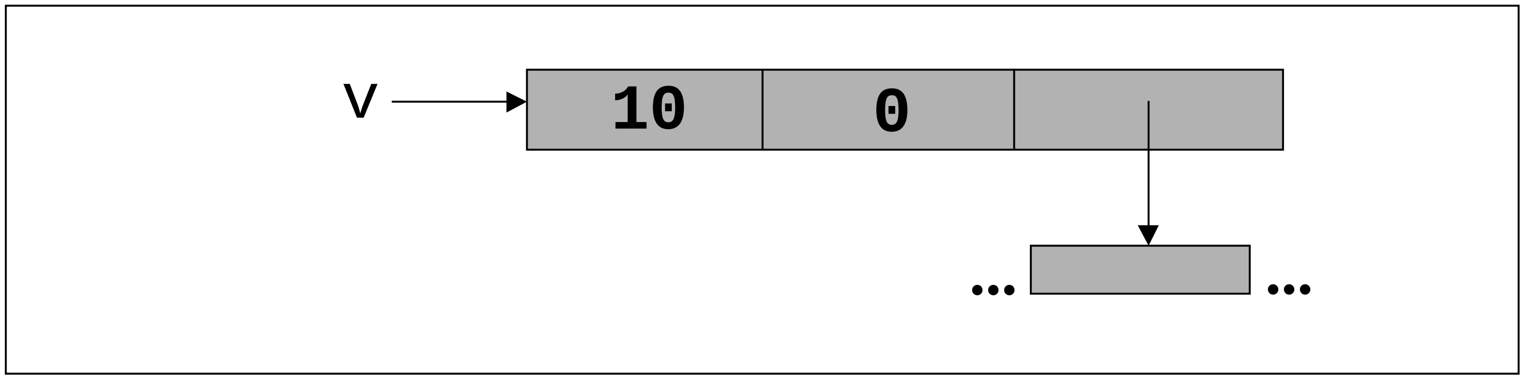
- How do I delete them?

```
delete myList;
delete [] myList;
```

EC++ 7. Check the return value of `new`

```
Vector(int size)         // specify size of vector
        : myCapacity(size),
          mySize(0),
          myList(new Item[size])
// postcondition: vector of size items constructed
{
   assert(myList != 0);
}
```

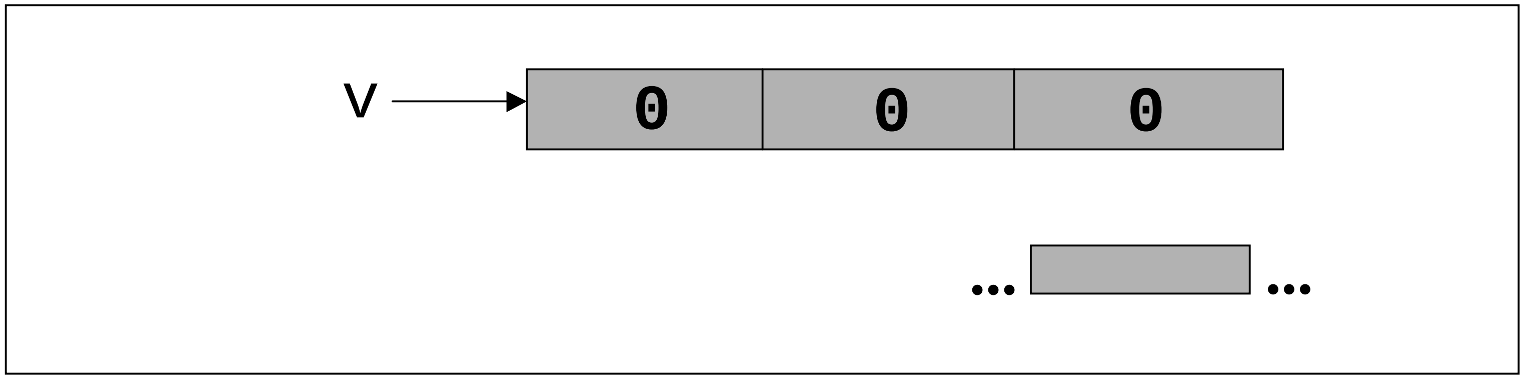
```
Vector<foo> * v = new Vector<foo>(10);
```



EC++ 6. Call `delete` on pointer members in destructors

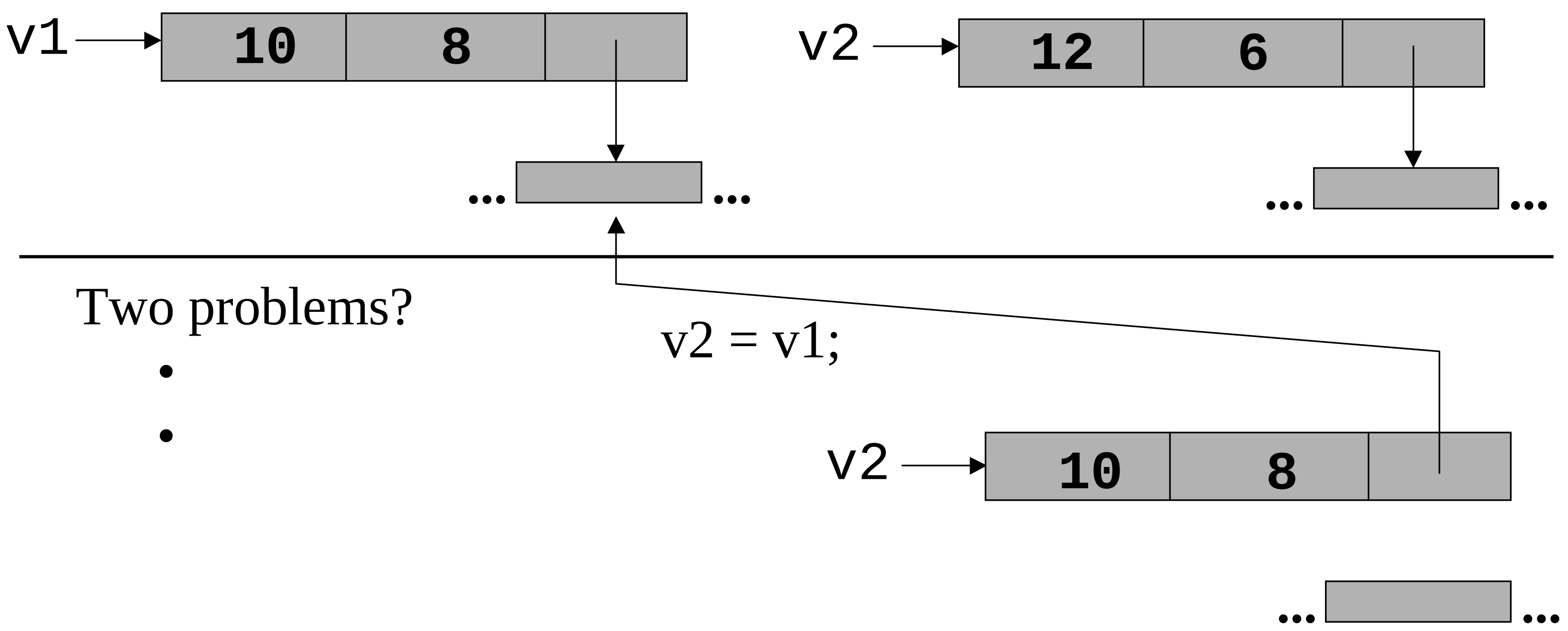
```
~Vector ()            // free new'd storage
// postcondition: dynamically allocated storage freed
{
// delete [] myList;
   myList = 0;
   myCapacity = 0;       // leave in "empty" state
   mySize = 0;
}
```

```
delete v;
```



EC++ 11. Define a copy constructor and an assignment operator for classes with dynamically allocated memory

- There default implementations will hurt you...



Two problems?

-
-

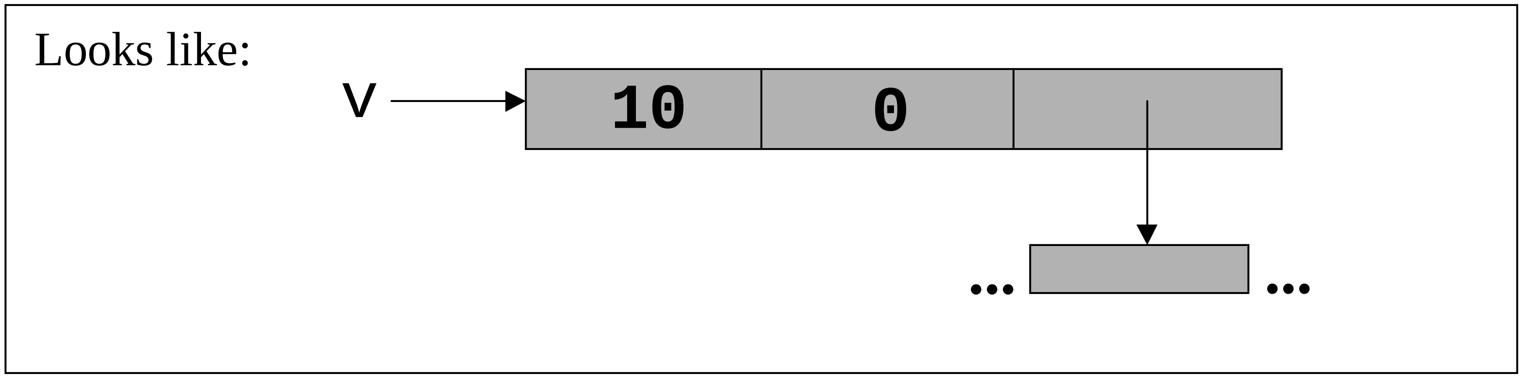
EC++ 8. Adhere to convention when writing `new`

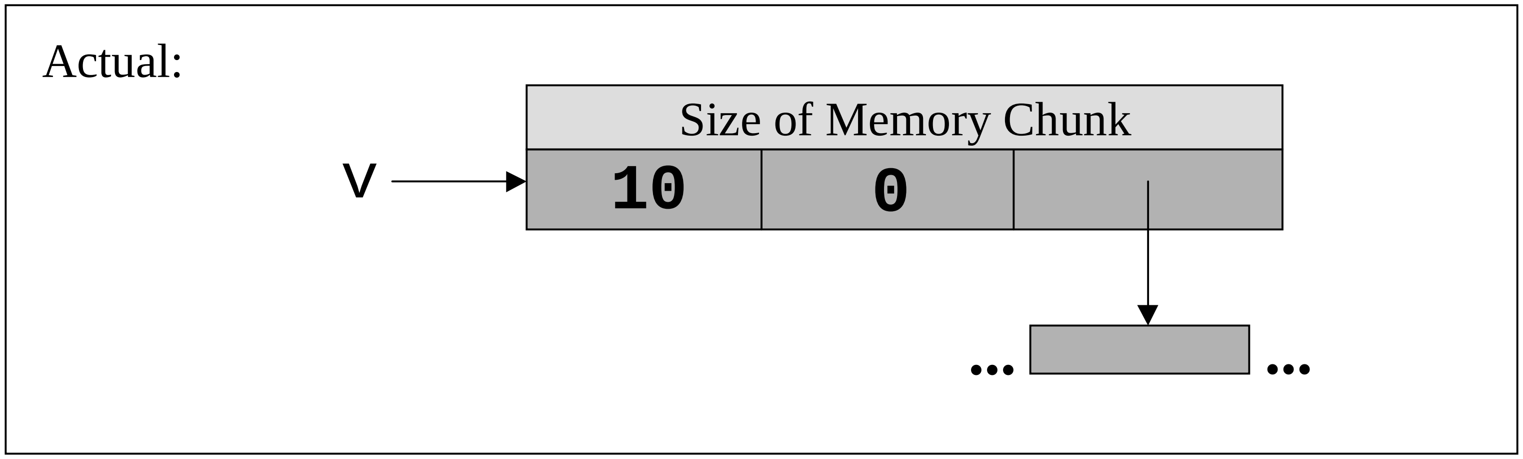
- You can overload `new`, just like any other operator.
  - You allocate your own memory
  - Return the right value
  - Call an error handling function when memory is not available

- Why is inheritance an issue here?

EC++ 10. Write `delete` if you write `new`

```
Vector<foo> * v = new Vector<foo>(10);
```





EC++ 12. Prefer initialization to assignment in constructors

- Why not?

```
Vector(int size)          // specify size of vector
   // postcondition: vector of size items constructed
{
   myCapacity = size;
   mySize = 0;
   myList = new Item[size];

   assert(myList != 0);
}
```

- const and reference members must be initialized, they can’t be assigned.

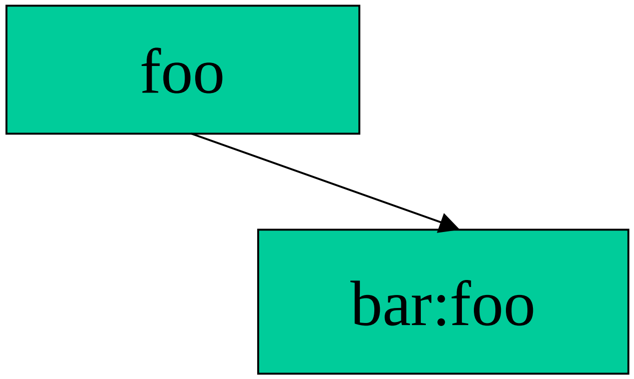
EC++ 13. List members in an initialization list in the order in which they are declared

```
private:
    int myCapacity;  // the capacity of the vector
    int mySize;      // # of elements in vector
    Item * myList;   // the array of items
```

- What happens?

```
Vector(int size)         // specify size of vector
       : myList(new Item[size]) ,
         myCapacity(myList.size()),
         mySize(0);
// postcondition: vector of size items constructed
{
   assert(myList != 0);
}
```

EC++ 14. Make destructors virtual in base classes



```
foo * f = …;     // What kind of objects
delete f;
```

Is Vector a good base class?

EC++ 15. Have `operator=` return a reference to `*this`

```
Vector & operator = (const Vector<Item> & vec) // overload =
// precondition: Item supports assignment
// postcondition: self is assigned vec
{
   // … LOTS OF CODE SNIPPED...
   return *this;
}
```

Mathematical operators support:

```
    w = x = y = z = 2;
```

So should your classes…

```
    w = x = y = z = Vector(10);
```

EC++ 16. Assign to all data members in `operator=`

```
Vector & operator = (const Vector<Item> & vec) // overload =
{
    // LOTS SNIPPED
    myList = new Item [myCapacity = vec.myCapacity];
    mySize = vec.mySize;
    // LOTS SNIPPED
}
```

- No kidding? Of course you must. Why is this hard?

EC++ 17. Check for assignment to self in `operator=`

```
Vector & operator = (const Vector<Item> & vec) // overload=
{
   if (this != &vec)      // don’t assign to self!
   {
        delete [] myList;    // out with old list, in with new
        // LOTS SNIPPED
   }
   return *this;
}
```

Is this legal?

`v = v;`