<table>
<thead>
<tr>
<th>Function</th>
<th>Prototype</th>
<th>Job</th>
<th>When is it called?</th>
<th>Default behavior if not defined</th>
<th>Reminders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constructor</td>
<td>ClassName();</td>
<td>Allocate memory for the object; declare object itself and member variables</td>
<td>Default is called when object is declared if no parameters are given. Nondefault is called if parameters are given.</td>
<td>Will declare all variables with garbage values, will not set up pointers. (Pointers will exist, but point to NULL)</td>
<td>If any constructor is defined then the compiler will not provide one, even if a default constructor is not defined.</td>
</tr>
<tr>
<td></td>
<td>ClassName (w/params);</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Copy Constructor</td>
<td>ClassName (const ClassName &amp;);</td>
<td>Copies the contents of the passed in object to the destination object.</td>
<td>1. Pass by value 2. Return value 3. When initializing an object with this constructor</td>
<td>Shallow copy: copy over the values stored in each variable Pointers point to same place as original!</td>
<td>Works with objects that can be assumed to be unitialized. (Don’t have to say new twice)</td>
</tr>
<tr>
<td>Function</td>
<td>Prototype</td>
<td>Job</td>
<td>When is it called</td>
<td>Default behavior if not defined</td>
<td>Reminders</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
<td>------------------------------------------</td>
<td>-----------------------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Assignment Operator</td>
<td>ClassName &amp; (const ClassName &amp;);</td>
<td>Copies the contents of the right operand to the left operand.</td>
<td>When setting an object of the same class type to another object of the same class type</td>
<td>Shallow copy, will only copy over the values stored in each variable</td>
<td>Left operand may already be initialized, must check for a delete preexisting memory when copying (Seriously, don’t say new twice)</td>
</tr>
<tr>
<td>Overload</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destructor</td>
<td>~ClassName();</td>
<td>Destroys the object</td>
<td>Any time an object goes out of scope</td>
<td>Will delete member variables, pointers to member objects</td>
<td>Define a destructor whenever there is dynamic memory</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1. When a function ends</td>
<td></td>
<td>(movie objects are members in instance of genre class) in order to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2. When the program ends</td>
<td></td>
<td>delete the member objects as well as the pointers to them</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3. A block containing a local variable ends</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>4. A delete operator is called</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Creating an Array Class

#include <algorithm>

class Array
{
private:
    int size;
    int* vals;

public:
    ~Array();
    Array( int s, int* v );
    Array( const Array& a );
    Array& operator=( const Array& a );
};
private:
    int size;
    int* vals;

public:
    ~Array();
};

Array::Array( int s, int* v )
{
    size = s;
    vals = new int[ size ];
    std::copy( v, v + size, vals );
}
Copy Constructor

```cpp
private:
    int size;
    int* vals;

public:
    ~Array();
};

Array::Array( const Array &a )
{
    size = a.size;
    vals = new int[ a.size ];
    std::copy( a.vals, a.vals + size, vals );
}
```
Assignment (=) Operator

Array& Array::operator =(const Array &a)
{
    if ( &a != this )
    {
        size = a.size;
        vals = new int[ a.size ];
        std::copy( a.vals, a.vals + size, vals );
    }
    return *this;
}
Destructor

private:
    int size;
    int* vals;

public:
    ~Array();
};

Array::~Array()
{
    delete vals;
    vals = NULL;
}
Using The Big 3

```cpp
int main()
{
    int vals[4] = {1, 2, 3, 4};
    Array a1(4, vals);
    Array a2(a1);
    a1 = a2;
    return 0;
}
```
GET A FLU SHOT