COMSC-200 Lecture Topic 9
OOP Arrays

Reference
Deitel Ch.11.9

The Array Class
an array of `int`s -- *not exactly Deitel's*
desired declaration: `Array a(10);`

data members
- `const int SIZE;`
- dynamically allocated

constructors
- specify array size -- *or not*
- because of dynamic memory, write:
  - a destructor, `~Array();`
  - a copy constructor, `Array(const Array&);`
  - an assignment operator, `Array& operator=(const Array&);`

operators
- stream insertion, with delimiter
- comparisons
- subscript -- `operator[]`
  - setter or getter?

`Array& operator=`
req'd due to dynamic memory allocation in class data members

`bool operator==`
and `bool operator!=`
`const` functions

`int&` *or* `int operator[]`
returning `const` *or not*
returning copy or original

`const` Data Members and `operator=`
because Array has const data member *and* assignment may be used:

```
Array& Array::operator=(const Array& array)
{
  if (this == &array) return *this;

  SIZE = array.SIZE; // CANNOT DO THIS
  ...

  return *this; // mutable self reference
}
```

there's one other way to do this...
- *if you can figure it out, you can use it*

The `const_cast` Operator
needed in lab 10
Solves "const" limitations in operator overloading
This will not work...

```
ID = rider.ID;
```

...indirectly with a pointer fails, too...

```
int* pID = &ID;
*pID = rider.ID;
```

...but this will work!

```
int* pID = const_cast<int*>(&ID);
*pID = rider.ID;
```

...so will this

```
int& rID = const_cast<int&>(ID);
rID = rider.ID;
```

...and either of these

```
*(const_cast<int*>(&ID)) = rider.ID; // or...
const_cast<int&>(ID) = rider.ID;
```

For pointers, use (for example, const char const name;):*

```
const_cast<const char*&>(name) = rider.name;
```

usually bad practice to use const-cast
- not okay to "force things to work"
- but okay in `operator=`
- and okay if you *know why you are using it*

to confirm

```
if(ID!=rider.ID)cout<<"ERROR";
if(strcmp(name,rider.name)!=0)cout<<"ERROR";
```

The New C++ Cast Operators
informit.com article

The `Elevator` Class
an empty elevator
inline functions (define *inside* class definition)
non-inline functions (define *outside* class definition)
Right-click here to download Elevator.exe
Right-click here to download VisualElevator.exe

Operator Overloading Key

Operator Type	Function Type	Return Type
comparison (`operator==`, etc)	getter member, friend, or stand-alone	`bool`
arithmetic (`operator+`, etc)	getter member, friend, or stand-alone	new object copy
assignment (`operator=`)	setter member	mutable self reference

compound assignment (`operator+=`, etc)	setter member	mutable self reference
type conversion (`operator int()`, etc)	getter member	*none*, but do return matching value
stream insertion (`operator<<`)	friend or stand-alone	mutable `ostream` reference
array subscript* (`operator[]`)	getter member	const reference to member data item
array subscript* (`operator[]`)	setter member	mutable reference to member data item

*Array subscript operators can be written as setters and/or getters. If both are written, the compiler determines which to apply based on the context in which it use used. So it uses the setter version if it can, and the getter version if it has to (like when using a const host object).

Ref: C++ Operators, Wiki
Ref: Operator Overloading Guidelines, CalTech