COMSC-200 Lecture Topic 10
The `vector` Class

□ Reference
Deitel Ch.11.11-11.13

□ Overloading `++` and `--`
pre- and post- increment/decrement
the dummy int parameter (post-increment/decrement)
- pre-increment: `operator++()`
- post-increment: `operator++(int)`
- pre-decrement: `operator--()`
- post-decrement: `operator--(int)`

pre: return object reference
post: return a *before* copy

□ A Timer Class
one-`int` member variable (not `const`)
`operator++` and `operator--` mutators
`operator+=` mutator
`operator<<` friend *OR* conversion operator

□ The STL `vector` Class
a generic array or linked-list, of:
- values (including objects!)
- pointers (zero values possible)

□ The STL `vector` Class *(cont)*
a templated class
- our Array class was for `int`s
- converting it to a template class...

`#include <vector>`
`using std::vector;`
declaration: {data type} {variable name}
- "array version": `vector<int> a(10);`
- "linked-list version": `vector<int> b;`

use as:
- local variable
- parameter, by val or by ref
- return value, by val or by ref
- private data member

`const` vectors as parameter or return
accessing and setting values
function library
traversal with `for`

□ `vector` 2-Parameter Constructor
2nd parameter is default object

□ `vector` Overloaded Assignment Operator
can copy a vector over another vector

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
pre-increment/decrement (`operator++()`)	setter member	mutable self reference
post-increment/decrement (`operator++(int)`)	setter member	copy of original

How to declare and fill a vector (as local variable):

```
vector<Time> v; // empty "linked list"
...
Time t(...);
v.push_back(t); // add an object
```

```
vector<Time> v(10); // "array" of 10 default objects
...
Time t(...);
v[0] = t; // replace the zeroth object
```

How to traverse a vector with a variable mutating pointer:

```
vector<Time> v; // a vector of Time objects
...
vector<Time>::iterator i;
for (i = v.begin(); i != v.end(); i++)
{
  ...
  ...  *i  ...
  ...
}
```

How to traverse a vector with an index:

```
int i;
for (i = 0; i < v.size(); i++)
{
  ...
  ...  v[i]  ...
  ...
}
```

How to traverse a vector with a variable read-only pointer:

```
vector<Time> v; // a vector of Time objects
...
vector<Time>::const_iterator i;
for (i = v.begin(); i != v.end(); i++)
{
  ...
  ...  *i  ...
  ...
}
```

vector as private data member:

```
class Schedule
{
  private:
  vector<Time> startTimes;
  ...
}
```

vector as parameter:

```
void fun(const vector<Time>&);

void fun(const vector<Time>& v)
{
  ...
}
```

vector as return value:

```
vector<Time> fun();

vector<Time> fun()
{
  vector<Time> result; // a vector of Time objects
  ...
  return result;
}
```

```
vector<Rider> Elevator::removeRidersForDestinationFloor()
{
  // create empty vector of removed riders to be used as return value
  // if elevator has any riders
    // create an empty vector for riders who remain on elevator
    // traverse vector of current riders
      // if a rider's destination floor is same as elevator's destination...
        // add rider to vector of removed riders
      // else
        // add rider to vector of remaining riders
    // reassign elevator rider vector to vector of remaining riders
  // return vector of removed riders
}

void Elevator::addRiders(const vector<Rider>& riders)
{
  // traverse the parameter vector
    // if there is still room on the elevator
      // add the rider to the elevator's rider vector
}

void Elevator::setDestinationBasedOnRiders()
```

```
{
  // if there are no riders on the elevator
    // exit the function

  // set elevator's destination to the zeroth Rider's destination
  // traverse the vector of elevator riders
    // get the absolute value of the distance from the elevator to the rider's destination floor
    // get the absolute value of the distance from the elevator to the elevator's destination floor
    // if closer to the rider's destination than to the elevator's destination
      // set elevator's destination to the rider's destination
}
```