7.3: Vectors (created by professor Kiyoung Song)

Vectors are similar to arrays, with the exception of vectors having the ability to grow (and shrink) in capacity. We will be working with the vectors from the C++ STL, or the **S**tandard **T**emplate **L**ibrary. Vectors are part of the std namespace.

#include <vector>

Vectors use dynamically allocated arrays to store each element.

```
1) std::vector<base_type> vector_name;
```

example: `vector<int> v;`
This creates a vector "v" that stores integers with a current capacity of 0.

```
2) std::vector<base_type> vector_name(capacity);
```

example: `vector<Rat> r(10);`
This creates a vector "r" that stores Rats with a current capacity of 10.
Each slot will contain a Rat object initialized with the default constructor.

```
3) std::vector<base_type> vector_name(capacity, initial_value);
```

example: `vector<double> r(4, 2.5);`

This creates a vector "r" that stores doubles with a current capacity of 4.

Each slot contains a double with a value of 2.5.

To add an element to the array, use the **push_back()** function.

```
std::vector<int> v;
v.push_back(5);
```

The two lines of code will do the following:

1) *Creates a vector that stores integers named "v". Current capacity is 0.*
2) *v.push_back(5) will attempt to add 5 to the vector. Since the current capacity is 0, the vector will create a heap array with a size of 1 and add 5 to the array.*
3) *Increases the counter to the number of elements in the array.*

(Note: These notes will be referring to the capacity of the vector as the capacity of the heap array.)

Subsequent calls to push_back will repeat a cycle where:

1) if the capacity of the vector is full, it will create a new heap array with a larger capacity*, copy over the values, and delete the previous array.
2) Add the element to the heap.

* The exact amount of increase depends on the implementation. Commonly, the size is doubled.

The constant need to delete/create heap arrays, on top of C++ controlling the increased array size, can affect efficiency.

The **reserve** function allocates the heap capacity to a certain number of elements.

```
std::vector<int> v;
v.reserve(5);
```

"Reserves" a heap array with a capacity of at least 5 elements. The above is NOT the same as

```
std::vector<int>v(5);
// A heap array of size 5 with 5 elements created using the default constructor
```

Capacity vs size

The **capacity()** function returns the maximum capacity of the dynamic array.
The **size()** function returns the number of elements that are currently stored in the array.

```
std::vector<int> v;
v.reserve(5);
v.push_back(1);
std::cout << v.capacity() << std::endl;
std::cout << v.size() << std::endl;
```

The **resize** function changes the **size** of the vector.

```
std::vector<int> v;
v.resize(5);
```

If the current size is *less than* the resize argument, it will fill in the size by adding new elements using the default constructor.
If the current size is *greater than* the resize argument, it will only keep the elements of the argument size starting from the first index.
It will automatically increase the capacity if needed.

Operator[]

Elements in the vector are accessible as you would access elements in arrays.

```
std::vector<int> v;
v.reserve(3);
v.push_back(1);
v.push_back(2);
v.push_back(3);
for (int i = 0; i < v.size(); i++) {
     std::cout << v[i] << " ";
}
```

However, do not use the [] operator to directly add elements to the array. Use the push_back function to add elements. Use the [] operator to read or swap elements.