Queue

enqueue
Putting Data Into a Queue
One of the 5 Queue Specific Operations

The Queue Component

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Let's look at the *enqueue* operation

All C++ *container* components have an operation that allows the client to insert data into the container, for Queue this operation is *enqueue*

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

enqueue

The job of *enqueue* is to move the value stored in parameter *x* to the rear of *self* and to clear *x*

Note *enqueue*, moves the value into the queue, it does not copy it

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

enqueue's spec indicates that the outgoing value of *self* equals the incoming value of *self* concatenated from the right with the incoming value *x*

Also, the outgoing value of *x* has been cleared

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

enqueue is called in the client below and the lines following the call contain comments based on *enqueue*'s spec

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1, q2;
Integer y1;
// ...
// Suppose q1 = <3,88> and q2 = <10>
y1 = 5;
q2.enqueue(y1);
// self = #self * <#x>
// clears x
}
```

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Substitute:

- q2 for *self*
- y1 for *x*

This gives us

Example client:

```
{
1 typedef Queue1<Integer> IntegerQueue;
2 IntegerQueue q1, q2;
3 Integer y1;
4 // ...
5 // Suppose q1 = <3,88> and q2 = <10>
6 y1 = 5;
7 q2.enqueue(y1);
8 // q2 = #q2 * <#y1>
9 // clears y1
}
```

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Now substitute:

- <10> for *#q2*
- 5 for *#y1*

This gives us

Example client:

```
{
1 typedef Queue1<Integer> IntegerQueue;
2 IntegerQueue q1, q2;
3 Integer y1;
4 // ...
5 // Suppose q1 = <3,88> and q2 = <10>
6 y1 = 5;
7 q2.enqueue(y1);
8 // q2 = <10> * <5>
9 // clears y1
}
```

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Now evaluate the concatenation: <10> * <5>

Which equals: <10,5>

This gives us

Example client:

```
{
1 typedef Queue1<Integer> IntegerQueue;
2 IntegerQueue q1, q2;
3 Integer y1;
4 // ...
5 // Suppose q1 = <3,88> and q2 = <10>
6 y1 = 5;
7 q2.enqueue(y1);
8 // q2 = <10,5>
9 // clears y1
}
```

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Recall that *clears* means resetting the value of the variable back to its initial value based on the constructor's ensures clause

For Integer, the initial value is zero

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1, q2;
Integer y1;
// ...
// Suppose q1 = <3,88> and q2 = <10>
y1 = 5;
q2.enqueue(y1);
// q2 = <10,5>
// clears y1
}
```

enqueue

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
   void transferFrom (Queue1& source);
   Queue1& operator = (Queue1& rhs);
// Queue1 Specific Operations
   void enqueue (T& x);
      //! updates self
      //! clears x
      //! ensures: self = #self * <#x>
   void dequeue (T& x);
   void replaceFront (T& x);
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Substitute:

- y1 = 0 for *clears y1*

enqueue's spec allows us to reason that the outgoing values of q2 and y are:

- q2 = <10,5>
- y1 = 0

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1, q2;
Integer y1;
// ...
// Suppose q1 = <3,88> and q2 = <10>
y1 = 5;
q2.enqueue(y1);
// q2 = <10,5>
// y1 = 0
}
```