Queue

replaceFront
Swapping Data In & Out of 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 *replaceFront* operation

Some C++ *container* components have an operation that allows the client to swap an item out of the container and replace it with another, for Queue this operation is *replaceFront*

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

replaceFront

The job of *replaceFront* is to swap the value stored at the front of the queue with the value stored in parameter *x*

Note *replaceFront*, moves the values, it does not copy them

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

replaceFront's ensures clause indicates:

- The outgoing value of x is equal to the front item of #self (the incoming queue)
- The outgoing value of *self* equals the *#self* with the item at the front of *#self* replaced by *#x* (the incoming value in parameter x)

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

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

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// <x> is prefix of #self
// self = <#x> * #self[1, |#self|)
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Substitute:

- q1 for *self*
- y2 for *x*

This gives us

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// <y2> is prefix of #q1
// q1 = <#y2> * #q1[1, |#q1|)
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Now substitute:

- <2,5,7> for #q1
- 3 for #y2

This gives us

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// <y2> is prefix of <2,5,7>
// q1 = <3> * <2,5,7>[1,|<2,5,7>|)
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Evaluate: <y2> is prefix of <2,5,7>

Giving y2's outgoing value: y2 = 2

This gives us

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// y2 = 2
// q1 = <3> * <2,5,7>[1, |<2,5,7>|)
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Evaluate: <2,5,7>[1, |<2,5,7>|]
= <2,5,7>[1, 3]
= <5,7>

This gives: q1 = <3> * <5,7>

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// y2 = 2
// q1 = <3> * <5,7>
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Evaluate: q1 = <3> * <5,7>
= <3,5,7>

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// y2 = 2
// q1 = <3,5,7>
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

replaceFront's ensures clause allows us to reason that the outgoing values of y2 and q1 are:

- y2 = 2
- q1 = <3,5,7>

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
// y2 = 2
// q1 = <3,5,7>
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

Now examine *replaceFront*'s requires clause

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

replaceFront's requires clause indicates the incoming value of *self* must not be empty

We must check the client's call to *replaceFront* to make sure it satisfies *replaceFront*'s requires clause

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
q1.replaceFront(y2);
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

In the client below a comment containing the *replaceFront*'s requires clause has been inserted prior to the call to *replaceFront*

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
// self /= <>
q1.replaceFront(y2);
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Substitute:

- q1 for *self*

This gives us

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
// q1 /= <>
q1.replaceFront(y2);
}
```

replaceFront

```
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);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self
      //! and
      //! self = <#x> * #self[1, |#self|)
   T& front (void);
   Integer length (void);
private: // representation
   // ...
};
```

Now substitute:

- <2,5,7> for q1

This gives us

replaceFront's requires shows that the incoming queue q1 is not empty

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2 = 3;
// ...
// Suppose q1 = <2,5,7>
// <2,5,7> /= <>
q1.replaceFront(y2);
}
```