Queue

dequeue

Getting Data 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 *dequeue* operation

All C++ *container* components have an operation that allows the client to extract data from the container, for Queue this operation is *dequeue*

dequeue

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

The job of *dequeue* is to move the value stored at the front of the queue into parameter *x*

Note *dequeue*, moves the value, it does not copy it

dequeue

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

dequeue’s ensures clause indicates:

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

dequeue

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

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

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer y2;
// ...
// Suppose q1 = <7,33,18>
q1.dequeue(y2);
// <x> is prefix of #self
// self = #self[1, |#self|)
}
```

dequeue

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

   void replaceFront (T& x);

   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;
  // ...
  // Suppose q1 = <7,33,18>
  q1.dequeue(y2);
  // <y2> is prefix of #q1
  // q1 = #q1[1, |#q1|)
}
```

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

dequeue

Now substitute:

- <7,33,18> for #q1

This gives us

Example client:

```
{
 typedef Queue1<Integer> IntegerQueue;
 IntegerQueue q1;
 Integer y2;
 // ...
 // Suppose q1 = <7,33,18>
 q1.dequeue(y2);
 // <y2> is prefix of <7,33,18>
 // q1 = <7,33,18>[1, |<7,33,18>|)
}
```

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

dequeue

Evaluate: <y2> is prefix of <7,33,18>

Giving y2’s outgoing value: y2 = 7

This gives us

Example client:

```
{
 typedef Queue1<Integer> IntegerQueue;
 IntegerQueue q1;
 Integer y2;
 // ...
 // Suppose q1 = <7,33,18>
 q1.dequeue(y2);
 // y2 = 7
 // q1 = <7,33,18>[1, |<7,33,18>|)
}
```

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

dequeue

Evaluate: q1 = <7,33,18>[1, |<7,33,18>|]
= <7,33,18>[1, 3]
= <33,18>

Giving q1's outgoing value: q1 = <33,18>

This gives us

Example client:

```
{
  typedef Queue1<Integer> IntegerQueue;
  IntegerQueue q1;
  Integer y2;
  // ...
  // Suppose q1 = <7,33,18>
  q1.dequeue(y2);
  // y2 = 7
  // q1 = <33,18>
}
```

dequeue

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

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

- y2 = 7
- q1 = <33,18>

Example client:

```
{
 typedef Queue1<Integer> IntegerQueue;
 IntegerQueue q1;
 Integer y2;
 // ...
 // Suppose q1 = <7,33,18>
 q1.dequeue(y2);
 // y2 = 7
 // q1 = <33,18>
}
```

dequeue

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

Now examine *dequeue*’s requires clause

dequeue

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

   void replaceFront (T& x);

   T& front (void);

   Integer length (void);

private: // representation

   // ...

};
```

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

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

Example client:

```
{
  typedef Queue1<Integer> IntegerQueue;
  IntegerQueue q1;
  Integer y2;
  // ...
  // Suppose q1 = <7,33,18>
  q1.dequeue(y2);
}
```

dequeue

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

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

Example client:

```
{
  typedef Queue1<Integer> IntegerQueue;
  IntegerQueue q1;
  Integer y2;
  // ...
  // Suppose q1 = <7,33,18>
  // self /= <>
  q1.dequeue(y2);
}
```

dequeue

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

   void replaceFront (T& x);

   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;
  // ...
  // Suppose q1 = <7,33,18>
  // q1 /= <>
  q1.dequeue(y2);
}
```

dequeue

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

Now substitute:

- <7,33,18> for q1

This gives us

dequeue's requires clause allows us to reason that the incoming queue q1 is not empty

Example client:

```
{
 typedef Queue1<Integer> IntegerQueue;
 IntegerQueue q1;
 Integer y2;
 // ...
 // Suppose q1 = <7,33,18>
 // <7,33,18> /= <>
 q1.dequeue(y2);
}
```