Queue

length

Querying the Queue About Its Size

One of the 5 Queue Specific Operations

```
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
   // ...
};
```

The Queue Component

Let's look at the *length* operation

All C++ *container* components in have an operation that allows the client to determine the number of items stored in the container, for Queue this operation is *length*

```
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);
      //! restores self
      //! ensures: length = |self|

private: // representation

   // ...

};
```

length

The job of *length* is to return an integer that represents the number of items currently stored in the queue

```
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);
      //! restores self
      //! ensures: length = |self|

private: // representation

   // ...

};
```

length

length's ensures clause indicates:

- That the integer returned is equal to the length of *self*

```
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);
      //! restores self
      //! ensures: length = |self|

private: // representation

   // ...

};
```

length

restores self

- Is concise notation for: *self = #self*
- Without this concise notation, the ensures clause would be written as follows:

  ensures: length = |self| and
  self = #self

length

```
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);
      //! restores self
      //! ensures: length = |self|
private: // representation
   // ...
};
```

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

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer z;
// ...
// Suppose q1 = <137,18,100,2,44>
z = q1.length();
// z = length = |self|
// self = #self
}
```

length

```
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);
      //! restores self
      //! ensures: length = |self|
private: // representation
   // ...
};
```

Substitute:

- q1 for *self*

This gives us

Example client:

```
{
 typedef Queue1<Integer> IntegerQueue;
 IntegerQueue q1;
 Integer z;
 // ...
 // Suppose q1 = <137,18,100,2,44>
 z = q1.length();
 // z = length = |q1|
 // q1 = #q1
}
```

length

```
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);
      //! restores self
      //! ensures: length = |self|

private: // representation

   // ...

};
```

Now substitute:

- <137,18,100,2,44> for #q1

This gives us

Example client:

```
{
   typedef Queue1<Integer> IntegerQueue;
   IntegerQueue q1;
   Integer z;
   // ...
   // Suppose q1 = <137,18,100,2,44>
   z = q1.length();
   // z = length = |<137,18,100,2,44>|
   // q1 = <137,18,100,2,44>
}
```

length

```
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);
      //! restores self
      //! ensures: length = |self|

private: // representation

   // ...

};
```

Evaluate: |<137,18,100,2,44>|

This produces a 5, so *length* returns: 5

So variable z is assigned 5
And q1’s value remains unchanged

Example client:

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
Integer z;
// ...
// Suppose q1 = <137,18,100,2,44>
z = q1.length();
// z = 5
// q1 = <137,18,100,2,44>
}
```