# Queue

*clear*

One of the 5 Standard 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 *clear* operation

All C++ components will have *clear*

## clear

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
      //! replaces self
      //! ensures: self = <>
   ~Queue1();
   void clear (void);
      //! clears self
   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 job of *clear* is to reset *self* to its initial value

## The *clears* Parameter Mode

```
template <class T>
class Queue1

{

public: // Standard Operations

   Queue1();
      //! replaces self
      //! ensures: self = <>

   ~Queue1();

   void clear (void);
      //! clears self

   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 *clears* parameter mode tells us to examine the constructor's ensures clause in order to determine what should be done to *self*

If we did not use this shorthand notation, then *clear*'s ensures clause would be:

```
    //! ensures: self = <>
```

## clear

```
template <class T>
class Queue1

{

public: // Standard Operations

   Queue1();

   ~Queue1();

   void clear (void);
     //! clears self

   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

   // ...

};
```

In the client below, the controlling object in the call to *clear* is variable q1

A comment containing clear's ensures clause has been added

Recall that clears self is shorthand for writing self = <> in the ensures clause

*Example client:*

```
{
  typedef Queue1<Integer> IntegerQueue;
  IntegerQueue q1;
  // ...
  // Suppose q1 = <3,88,5>
  q1.clear();  // self = <>
}
```

```
template <class T>
class Queue1
{
public: // Standard Operations
   Queue1();
   ~Queue1();
   void clear (void);
     //! clears self
   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
   // ...
};
```

# clear

After substituting q1 for *self*, *clear's* spec allows us to reason that the outgoing value of q1 is reset to the empty string

*Example client:*

```
{
typedef Queue1<Integer> IntegerQueue;
IntegerQueue q1;
// ...
// Suppose q1 = <3,88,5>
q1.clear();   // q1 = <>
}
```