

Queue

transferFrom and *operator* =
Data Movement Operations
Two of the 5 Standard Operations

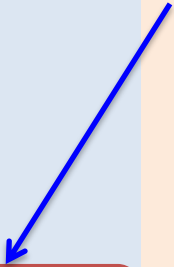
The Queue Component

Let's look at the data movement operations

All C++ components in this course will have these two data movement operations:

1. `transferFrom`
2. `operator =`

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



```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
        ///! replaces self
        ///! clears source
        ///! ensures self = #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
    // ...
};

```

transferFrom

The job of *transferFrom* is to move the value stored in parameter *source* to *self* and to clear *source*

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

transferFrom

```
template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
    //! replaces self
    //! clears source
    //! ensures self = #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
    // ...
};
```

transferFrom's spec indicates two things:

1) *self*'s incoming value will be *replaced* so that its output value will be equal to the incoming value of *source*

2) *source* will be *cleared* so that its outgoing value will be an initial value, i.e., an empty queue

Recall - a variable has two values:

- outgoing value – variable prepended with ‘#’
- incoming variable – variable is undecorated

transferFrom

```
template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
        //! replaces self
        //! clears source
        //! ensures self = #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
    // ...
};
```

transferFrom is called in the client below and the line following the call contains a comment based on *transferFrom*'s spec

Example client:

```
{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2.transferFrom(q1);
6  // clears source and self = #source
}
```

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
        //! replaces self
        //! clears source
        //! ensures self = #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
    // ...
};

```

transferFrom

Substitute:

- q2 for *self*
- q1 for *source*

This gives us

Example client:

```

{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2.transferFrom(q1);
6  // clears q1 and q2 = #q1
}

```

transferFrom

```
template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
        //! replaces self
        //! clears source
        //! ensures self = #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
    // ...
};
```

Now substitute:

- <3,88,5> for #q1

This gives us

Example client:

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

transferFrom

```
template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();
    void clear (void);
    void transferFrom (Queue1& source);
        //! replaces self
        //! clears source
        //! ensures self = #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
    // ...
};
```

Finally, replace *clears q1* by utilizing the constructor's ensures clause and obtain

transferFrom's spec allows us reason that q1's original value has been moved to q2, and that q1 has been reset to the initial value

Example client:

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



```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);

    Queue1& operator = (Queue1& rhs);
        //! replaces self
        //! restores rhs
        //! ensures: self = rhs

    // Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

operator = copies the value stored in *rhs* to *self* and it leaves *rhs* unchanged

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);

    Queue1& operator = (Queue1& rhs);
        //! replaces self
        //! restores rhs
        //! ensures: self = rhs

// Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

The first line below is an example showing *operator =* being called using normal *infix syntax*.

The 2nd line shows how the C++ compiler really views the call using *object-oriented syntax*:

```

q2 = q1;
↓
q2.operator = (q1);

```

In *operator =*'s spec, *self* refers to *q2*, which is the controlling object, and *rhs* refers to *q1*

Note – in this call to *operator =*

- The *actual parameter* is *q1*
- The corresponding *formal parameter* is *rhs*

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);

    Queue1& operator = (Queue1& rhs);
        //! replaces self
        //! restores rhs
        //! ensures: self = rhs

    // Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

operator = is called in the client below and the line following the call contains a comment based on *operator =*'s spec

Example client:

```

{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2 = q1;
6  // restores rhs and self = rhs
}

```

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);
    Queue1& operator = (Queue1& rhs);
    //! replaces self
    //! restores rhs
    //! ensures: self = rhs

// Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

Recall that *restores* means the outgoing value of the parameter equals the incoming value

So, *restores rhs* is equivalent to writing:
`rhs = #rhs`

Below, *restores rhs* has been replaced with
`rhs = #rhs`

Example client:

```

{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2 = q1;
6  // rhs = #rhs and self = rhs
}

```

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);
    Queue1& operator = (Queue1& rhs);
        //! replaces self
        //! restores rhs
        //! ensures: self = rhs

// Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

Substitute:

- q1 for *rhs*
- q2 for *self*

This gives us

Example client:

```

{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2 = q1;
6  // q1 = #q1 and q2 = q1
}

```

```

template <class T>
class Queue1
{
public: // Standard Operations
    Queue1();
    ~Queue1();

    void clear (void);
    void transferFrom (Queue1& source);
    Queue1& operator = (Queue1& rhs);
        //! replaces self
        //! restores rhs
        //! ensures: self = rhs

// Queue1 Specific Operations
    void enqueue (T& x);
    void dequeue (T& x);
    void replaceFront (T& x);
    T& front (void);
    Integer length (void);
private: // representation
    // ...
};

```

operator =

Now substitute:

- <3,88,5> for #q1

This gives us

operator ='s spec allows us to reason that the outgoing values of q1 and q2 are:

- q1 = <3,88,5>
- q2 = q1

Example client:

```

{
1  typedef Queue1<Integer> IntegerQueue;
2  IntegerQueue q1, q2;
3  // ...
4  // Suppose q1 = <3,88,5> and q2 = <10>
5  q2 = q1;
6  // q1 = <3,88,5> and q2 = q1
}

```