A Detailed Explanation Of the Sequence Component

Part 3

Adding/Removing/Replacing Values

The Sequence Component

```
template <class T>
class Sequence1
{
public: // Standard Operations
   Sequence1();
   ~Sequence1();
   void clear(void);
   void transferFrom(Sequence1& source);
   Sequence1& operator =(Sequence1& rhs);
// Sequence1 Specific Operations
   void add(Integer pos, T& x);
   void remove(Integer pos, T& x);
   void replaceEntry(Integer pos, T& x)
   T& entry(Integer pos);
   void append(Sequence1& sToApppend);
   void split(Integer pos,
            Sequence1& receivingS);
   Integer length(void);
private: // representation
   // ...
};
```

Three of the 7 *Sequence Specific Operations* have to do with adding, removing, or replacing values in the sequence

add

```
template <class T>
class Sequence1

{

public: // Standard Operations

   Sequence1();

   ~Sequence1();

   void clear(void);

   void transferFrom(Sequence1& source);

   Sequence1& operator =(Sequence1& rhs);

// Sequence1 Specific Operations

   void add(Integer pos, T& x);
      //! updates self
      //! restores pos
      //! clears x
      //! requires: 0 ≤ pos ≤ |self|
      //! ensures: self =
      //!  #self[0, pos) * <#x> *
      //!  #self[pos, |#self|)

   void remove(Integer pos, T& x);

   void replaceEntry(Integer pos, T& x)

   ...

   Integer length(void);

private: // representation

   // ...

};
```

The job of *add* is to move the value stored in parameter *x* into *self* at location *pos*

Note *add*, moves the value into the sequence, it does not copy the value

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <"C343","C251","C455">
// y = "B461" and k = 2
   s1.add(k,y);
// outgoing s1, y, and k
// s1 = <"C343","C251","B461","C455">
// y = "" and k = 2
```

add's requires clause

```
template <class T>
class Sequence1

{

public: // Standard Operations

   Sequence1();

   ~Sequence1();

   void clear(void);

   void transferFrom(Sequence1& source);

   Sequence1& operator =(Sequence1& rhs);

// Sequence1 Specific Operations

   void add(Integer pos, T& x);
      //! updates self
      //! restores pos
      //! clears x
      //! requires: 0 ≤ pos ≤ |self|
      //! ensures: self =
      //!  #self[0, pos) * <#x> *
      //!  #self[pos, |#self|)

   void remove(Integer pos, T& x);

   void replaceEntry(Integer pos, T& x)

   ...

   Integer length(void);

private: // representation

   // ...

};
```

add requires that the location to add the item designated by parameter *pos* be within the bounds of *self*

The client below is defective because the call to *add* violates the requires clause

Example:

```
typedef Sequence<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <"C343","C251","C455">
// y = "B438" and k = 4
   s1.add(k,y);
// outgoing s1, y, and k
// s1 = ???
// k = ???
// y = ???
```

remove

```
template <class T>
class Sequence1
{
public: // Standard Operations
   Sequence1();
   ~Sequence1();
   void clear(void);
   void transferFrom(Sequence1& source);
   Sequence1& operator =(Sequence1& rhs);
// Sequence1 Specific Operations
   void add(Integer pos, T& x);
   void remove(Integer pos, T& x);
      //! updates self
      //! restores pos
      //! replaces x
      //! requires: 0 ≤ pos < |self|
      //! ensures: <x> =
      //!  #self[pos, pos+1) and
      //!  self =
      //!  #self[0, pos) *
      //!  #self[pos+1, |#self|)
   void replaceEntry(Integer pos, T& x)
   ...
   Integer length(void);
private: // representation
   // ...
};
```

The job of *remove* is to move into parameter *x* the value stored at location *pos* in *self*

Note *remove*, moves the value out of the sequence, and it does not make a copy

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <"C343","C251","C455","B461">
// y = "A247" and k = 1
   s1.remove(k,y);
// outgoing s1, y, and k
// s1 = <"C343","C455","B461">
// y = "C251" and k = 1
```

```
template <class T>
class Sequence1

{

public: // Standard Operations

   Sequence1();

   ~Sequence1();

   void clear(void);

   void transferFrom(Sequence1& source);

   Sequence1& operator =(Sequence1& rhs);

// Sequence1 Specific Operations

   void add(Integer pos, T& x);

   void remove(Integer pos, T& x);
      //! updates self
      //! restores pos
      //! replaces x
      //! requires: 0 ≤ pos < |self|
      //! ensures: <x> =
      //!  #self[pos, pos+1) and
      //!  self =
      //!  #self[0, pos) *
      //!  #self[pos+1, |#self|)

   void replaceEntry(Integer pos, T& x)

   ...

   Integer length(void);

private: // representation

   // ...

};
```

remove's requires clause

remove requires that the location from which to remove the item, designated by parameter *pos*, be within the bounds of *self*

The client below is defective because the call to *remove* violates the requires clause

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <"C343","C251","C455","B461">
// y = "A247" and k = 4
   s1.remove(k,y);
// outgoing s1, y, and k
// s1 = ???
// k = ???
// y = ???
```

replaceEntry

```
template <class T>
class Sequence1
{
public: // Standard Operations
   Sequence1();
   ~Sequence1();
   void clear(void);
   void transferFrom(Sequence1& source);
   Sequence1& operator =(Sequence1& rhs);
// Sequence1 Specific Operations
   void add(Integer pos, T& x);
   void remove(Integer pos, T& x);
   void replaceEntry(Integer pos, T& x)
      //! updates self, x
      //! restores pos
      //! requires: 0 ≤ pos < |self|
      //! ensures: <x> =
      //!  #self[pos, pos+1) and
      //!  self = #self[0, pos) * <#x> *
      //!  #self[pos+1, |#self|)
   ...
   Integer length(void);
private: // representation
   // ...
};
```

The job of *replaceEntry* is twofold:

1. move the value stored at location *pos* in *self* out of *self* and into parameter *x*
2. move the incoming value of parameter *x* into *self* at location *pos*

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <"C343","C251","C455","B461">
// k = 2
// y = "A247"
   s1.replaceEntry(k,y);
// outgoing s1, y, and k
// s1 = <"C343","C251","A247","B461">
// k = 2
// y = "C455"
```

replaceEntry's requires clause

```
template <class T>
class Sequence1

{

public: // Standard Operations

   Sequence1();

   ~Sequence1();

   void clear(void);

   void transferFrom(Sequence1& source);

   Sequence1& operator =(Sequence1& rhs);

// Sequence1 Specific Operations

   void add(Integer pos, T& x);

   void remove(Integer pos, T& x);

   void replaceEntry(Integer pos, T& x)
      //! updates self, x
      //! restores pos
      //! requires: 0 ≤ pos < |self|
      //! ensures: <x> =
      //!  #self[pos, pos+1) and
      //!  self = #self[0, pos) * <#x> *
      //!  #self[pos+1, |#self|)

   ...

   Integer length(void);

private: // representation

   // ...

};
```

replaceEntry requires that the location to replace the item, designated by parameter *pos*, be within the bounds of *self*

The client below is defective because the call to *replaceEntry* violates the requires clause

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1;
Text y;
Integer k;
...
// incoming s1, y, and k
// s1 = <>
// y = "B481" and k = 0
   s1.replaceEntry(k,y);
// outgoing s1 and y
// s1 = ???
// k = ???
// y = ???
```