A Detailed Explanation Of the Sequence Component

Part 5

append and split

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
   // ...
};
```

Two of the 7 *Sequence Specific Operations* have to do with manipulating two Sequences at the same time

append

```
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);
      //! updates self
      //! clears sToAppend
      //! ensures: self =
      //!  #self * #sToAppend
   void split(Integer pos,
             Sequence1& receivingS);
   Integer length(void);
private: // representation
   // ...
};
```

The job of *append* is to allow the client program to concatenate together two sequence variables

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1, s3;
...
// incoming s1 and s3
// s1 = <"C343","C251","C455">
// s3 = <"red","blue">
s3.append(s1);
// outgoing s1 and s3
// s1 = <>
// s3 = <"C343","C251","C455","red","blue">
```

split

```
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 append(Sequence1& sToApppend);
   void split(Integer pos,
              Sequence1& receivingS);
      //! updates self
      //! restores pos
      //! replaces receivingS
      //! requires: 0 ≤ pos ≤ |self|
      //! ensures: self =
      //!     #self[0, pos) and
      //!  receivingS =
      //!     #self[pos, |#self|)
   Integer length(void);
private: // representation
   // ...
};
```

The job of *split* is to allow the client program to break apart a sequence at location *pos*

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1, s3;
Integer k;
...
// incoming s1, s3, and k
// s1 = <"C343","C251","C455">
// s3 = <"red","blue","yellow","purple","green">
// k = 2
s3.split(k,s1);
// outgoing s1, s3, and k
// s1 = <"yellow","purple","green">
// s3 = <"red","blue">
// k = 2
```

split'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 append(Sequence1& sToApppend);
   void split(Integer pos,
              Sequence1& receivingS);
      //! updates self
      //! restores pos
      //! replaces receivingS
      //! requires: 0 ≤ pos ≤ |self|
      //! ensures: self =
      //!     #self[0, pos) and
      //!  receivingS =
      //!     #self[pos, |#self|)
   Integer length(void);
private: // representation
   // ...
};
```

split requires that the location specified parameter *pos* to break apart *self* by be within the bounds of *self*

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

Example:

```
typedef Sequence1<Text> TextSeq;
TextSeq s1, s3;
Integer k;
...
// incoming s1, s3, and k
// s1 = <"C343","C251","C455">
// s3 = <"red","blue","yellow","purple","green">
// k = 10
s3.split(k,s1);
// outgoing s1, s3, and k
// s1 = ???
// s3 = ???
// k = ???
```