CS 132 - Spring 2005
last modified: 4-06-05

Initial "UML" for **binary_tree** template class (last modified: 4-06-05)

adapted from Ch. 10, Savitch and Main, "Data Structures and Other Objects Using C++"

Template Class: `binary_tree<Item>`

/* a binary tree where each node contains an Item */

Member data and related details:

* contains elements of type **`value_type`**; this is set to be the value of template parameter **Item**

* has a size of **`size_t`**

* Each non-empty binary_tree instance always has a "current node". The location of the current node is controlled by three member functions: shift_to_root, shift_left, and shift_right.

Constructors:

/* postcondition: creates an empty **binary_tree** instance (with no nodes) */

```
binary_tree( );
```

Accessors and other constant member functions:

/* postcondition: returns the number of nodes in the binary_tree. */

```
size_t       get_size( )       const;
```

/* postcondition: returns **true** if binary_tree is empty, and returns **false** otherwise */

```
bool         is_empty( ) const;
```

/* precondition: size( ) > 0 */

/* postconditions: returns the data from the "current node", BUT the binary_tree is unchanged. */

```
Item         retrieve( ) const;
```

/* postcondition: returns true if size( ) > 0 and the "current node" is the root */

```
bool         is_root( ) const;
```

/* postcondition: returns true if size( ) > 0 and the "current node" is a leaf (has no children) */

```
bool         is_leaf( ) const;
```

/* postcondition: returns true if size( ) > 0 and the "current node" has a parent */

```
bool         has_parent( ) const;
```

/* postcondition: returns true if size( ) > 0 and the "current node" has a left child */

```
bool         has_left_child( ) const;
```

/* postcondition: returns true if size( ) > 0 and the "current node" has a right child */

```
bool         has_right_child( ) const;
```

Modifiers and other modifying member functions:

/* precondition: size( ) == 0 */

/* postconditions: the binary_tree now has one node (a root node) containing the specified entry. The new root node is the "current node". */

```
void         create_root(const Item& entry);
```

```
/*    preconditions: size( ) > 0, and has_left_child( ) == false */
/*    postcondition: a left child has been added to the "current node", with the given entry as its value */
void          add_left(const Item& entry);

/*    preconditions: size( ) > 0, and has_right_child( ) == false */
/*    postcondition: a right child has been added to the "current node", with the given entry as its value */
void          add_right(const Item& entry);

/*    preconditions: size( ) > 0, and has_left_child( ) == false */
/*    postcondition: a left subtree has been added to the "current node", with the given tree as its value */
void          add_left_subtree(binary_tree<Item>& left_subtree);

/*    preconditions: size( ) > 0, and has_right_child( ) == false */
/*    postcondition: a right subtree has been added to the "current node", with the given tree as its value */
void          add_right_subtree(binary_tree<Item>& right_subtree);

/*    precondition: size( ) > 0 */
/*    postcondition: The data at the "current node" has been changed to the new entry */
void          change(const Item& entry);

/*    preconditions: size( ) > 0, and has_left_child( ) == true */
/*    postcondition: the left subtree of the current node has been removed from the tree. */
void          remove_left_subtree( );

/*    preconditions: size( ) > 0, and has_right_child( ) == true */
/*    postcondition: the right subtree of the current node has been removed from the tree. */
void          remove_right_subtree( );

/*    postconditions: the tree is empty (and so there is no "current node", either) */
void          clear_tree( );

/*    precondition: size( ) > 0 */
/*    postcondition: the "current node" is now the root of the tree. */
void          shift_to_root( );

/*    precondition: has_left_child( ) == true */
/*    postcondition: the "current node" has been shifted down to the left child of the old current node. */
void          shift_left( );

/*    precondition: has_right_child( ) == true */
/*    postcondition: the "current node" has been shifted down to the right child of the old current node. */
void          shift_right( );

/*    preconditions: if !empty( ), depth is the depth of the calling binary_tree instance. */
/*    postconditions: if  !empty, then the contents of the root and all of its descendants have been written to
      cout with the << operator using a backward in-order traversal. Each node is indented four times its
      depth. */
void print_tree(size_t depth);
```