ActionScript 2.0 Library
ASLib

http://www.springsoft.org

Manual
v1.1.0

{amsler}@users.sourceforge.net

January 22, 2005

Introduction:

ASLib is an ActionScript 2.0 based library that implements common data structures. This release contains the following data structures:

- Single Linked List
- Stack
- Queue
- Hash Table
- Binary Tree

Description:

Single Linked List
The list is composed of list nodes, which can hold any data type in form of a Single Linked List Object. Default inserts are made at the head of the list. Nodes in the list have unique keys that can be used for data searches and or node removal.

Stack:
The stack uses the single linked list for its internal structure. The stack is a regular “last-in, first-out”, LIFO, structure. It can hold any data type in form of a StackObject.

Queue:
The queue uses the single linked list for its internal structure. The queue is a regular “first-in, first-out”, FIFO, structures. It can hold any data type in form of a QueueObject.

HashTable:
The hash table uses an array of single linked lists. It can hold any data type in form of a HashTableObject. The HashTableObject’s numeric key is used to hash to a linked list array location using the modulus on the object key and the array length. Once the single linked list array location is determined, a default insert at the front of the list is performed. The object key is not only used to determine the correct array hash location but also to find the object within a linked list while performing a search and or a removal of an object.

BinaryTree:
The binary tree can hold any data type in from of a BinaryTreeObject that is wrapped in a Tree Node object that has references to its left and right tree node siblings. This implementation has recursive as well as iterative tree node insert methods.

API:

Please refer to our online documentation at “http://www.springsoft.org” for detailed API listings.

Usage:

The following User class is used in all examples as the data object. The only difference amongst all User classes is the interface class which the User class implements.

```
class User implements SingleLinkedListObject  {

        function User(key:Number, firstName:String, lastName:String) { ... }

        public function getKey():Number { ... }

        public function toString():String { ... }

        public function getFirstName():String { .. }

        public function getLastName():String { .. .}
```

Single Linked List:

```
import org.springsoft.aslib.SingleLinkedList;

// Create single linked list
singleLinkedList = new SingleLinkedList();

// Create some user objects
var u1 = new User(1, "FN " + 1, "LN " + 1);
var u2 = new User(2, "FN " + 2, "LN " + 2);
var u3 = new User(3, "FN " + 3, "LN " + 3);
var u4 = new User(4, "FN " + 4, "LN " + 4);

// Insert user objects into list
singleLinkedList.insert(u1);
singleLinkedList.insert(u2);
singleLinkedList.insert(u3);
singleLinkedList.insert(u4);

// Print the list
singleLinkedList.print();

// Get the size of the list
var listSize = singleLinkedList.size();
trace("SingleLinkedList size = " + listSize);

// Remove a node
singleLinkedList.remove(u1.getKey());

// Print the list
singleLinkedList.print();

// Getting a user object identified by a key
// First option: Get the ListNode and then call get() on the ListNode
// to get the data object (SingleLinkedList object)
trace("\nGetting user object with key: " + singleLinkedList.get(u2.getKey()).get());
// Second option: Get the data object (SingleLinkedList object) directly
trace("Getting user object with key: " + singleLinkedList.getData(u2.getKey()));

// Getting a user object identified by an index where index in {0, ..., n-1}
// The index starts at 0 just like an array does
// First option: Get the ListNode and then call get() on the ListNode
// to get the data object (SingleLinkedList object)
trace("Getting user object with index: " + singleLinkedList.getAt(1).get());
// Second option: Get the data object (SingleLinkedList object) directly
trace("Getting user object with index: " + singleLinkedList.getDataAt(1));

// Remove remaining nodes and print result
singleLinkedList.removeAt(2);
singleLinkedList.print();
singleLinkedList.removeAt(1);
singleLinkedList.print();
singleLinkedList.remove(u4.getKey());
singleLinkedList.print();
```

Stack:

```
import org.springsoft.aslib.Stack;

// Create a stack object
var stack = new Stack();

// Create some user objects
var u1 = new User(1, "FirstName 1”, "LastName 1”);
var u2 = new User(2, "FirstName 2”, "LastName 2”);
var u3 = new User(3, "FirstName 3”, "LastName 3”);
var u4 = new User(4, "FirstName 4” "LastName 4”);

// Push some user objects onto stack
stack.push(u1);
stack.push(u2);
stack.push(u3);

// Print stack
stack.print();

// Pop stack
stack.pop();

// Print stack
stack.print();

// Peek stack
stack.peek();

// Print stack
stack.print();

// Pop stack
stack.pop();

// Push user object onto stack
stack.push(u4);
stack.print();

// Poping remaining objects and printing stack
stack.pop();
stack.print();

stack.pop();
stack.print();
```

Queue:

```
import org.springsoft.aslib.Queue;

// Create queue instance
var queue = new Queue();

// Create some user objects
var u1 = new User(1, "FirstName 1”, "LastName 1”);
var u2 = new User(2, "FirstName 2”, "LastName 2”);
var u3 = new User(3, "FirstName 3”, "LastName 3”);
var u4 = new User(4, "FirstName 4” "LastName 4”);

// Put users objects into queue
queue.enqueue(u1);
queue.enqueue(u2);
queue.enqueue(u3);

// Print queue
queue.print();

// Remove user object from queue
trace("\nQueue.dequeue() = " + queue.dequeue());

// Remove user object from queue
trace("\nQueue.dequeue() = " + queue.dequeue());

// Print queue
queue.print();

// Put user object into queue
queue.enqueue(u4);

// Print queue
queue.print();

// Remove user object from queue
trace("\nQueue.dequeue() = " + queue.dequeue());

// Print queue
queue.print();

// Remove user object from queue
trace("\nQueue.dequeue() = " + queue.dequeue());

// Print queue
queue.print();
```

Hash Table:

```
import org.springsoft.aslib.HashTable;

// Create HashTable of size 20
var hashTable = new HashTable(20);

// Check if HashTable is empty
trace("Is HashTable empty = " + hashTable.isEmpty());

// Create some user objects
var u1 = new User(1, "FirstName 1”, "LastName 1”);
var u2 = new User(2, "FirstName 2”, "LastName 2”);
var u3 = new User(3, "FirstName 3”, "LastName 3”);
var u4 = new User(4, "FirstName 4” "LastName 4”);

// Put the created user objects in the HashTable
hashTable.put(user1.getKey(), user1);
hashTable.put(user2.getKey(), user2);
hashTable.put(user3.getKey(), user3);
hashTable.put(user4.getKey(), user4);

// Check if HashTable is empty
trace("\nIs HashTable empty = " + hashTable.isEmpty());

// Print HashTable size
trace("\nHashTable size = " + hashTable.size());

// Access an HashTable object and print it
trace(hashTable.get(4567).toString());

// Print the HashTable
hashTable.print();

// Remove an object from the HashTable
hashTable.remove(4567);

// Print HashTable size
trace("\nHashTable size = " + hashTable.size());

// Print HashTable
hashTable.print();

// Clear HashTable
hashTable.removeAll();

// Print HashTable
hashTable.print();
```

Binary Tree:

```
import org.springsoft.aslib.BinaryTree;

// Create a binary tree object
var binaryTree = new BinaryTree();

// Inserting some User objects using both
// the recursive and iterative insertion mehtod
binaryTree.insert(new User(100, "FirstName 100", "LastName 100”));
binaryTree.insertIter(new User(80, "FirstName 80", "LastName 80”));
binaryTree.insert(new User(120, "FirstName 120”, "LastName 120"));
binaryTree.insertIter(new User(110, "FirstName 110", "LastName 110"));
binaryTree.insert(new User(70, "FirstName 70", "LastName 70"));
binaryTree.insertIter(new User(75, "FirstName 75", "LastName 75"));
binaryTree.insert(new User(130, "FirstName 130", "LastName 130"));
binaryTree.insertIter(new User(60, "FirstName 60", "LastName 60"));
binaryTree.insert(new User(90, "FirstName 90", "LastName 90"));

// Print the tree in PreOrder
binaryTree.printPreOrder();

// Print the tree in InOrder
binaryTree.printInOrder();

// Print the tree in PostOrder
binaryTree.printPostOrder();

// Search the tree with key and print the object
trace("Searching(70): " + binaryTree.search(70).toString());

// Remove a node
binaryTree.remove(80);

// Print tree in PreOrder
binaryTree.printPreOrder();
```