## OM Test Plan and Test Cases/Applications

### Symbol Table Test Plans

#### Red Black Tree implementation

Various scenarios have been enumerated here.

1. **Functional Tests** – test for the primary functionality of the data type.

* 1. Create / Destroy Tests – these are for testing various constructors & the destructor.
     1. Automatic Tree
     2. Dynamic Tree
  2. Insertion Tests
     1. Normal (Common) Cases – without duplicates
        + Tree arising from Arbitrary Sequence of 5 Insertions
        + Tree arising from Arbitrary Sequence of 10 Insertions
        + Tree arising from Arbitrary Sequence of 15 Insertions
     2. Duplicate Insertion (3 cases)
  3. Find Tests
     1. Existing Keys (5 cases)
     2. Missing Keys (5 cases)
  4. Print Tests
  5. Traversal Tests / Sorting Order
  6. Visitor Tests

1. **API Tests** – test all APIs for the data type (library). All variants for an API (depending on default parameters) need to be tested.
   1. Create – wraps the constructor
   2. Insert
   3. Find
   4. Validate
   5. Print
   6. Release – wraps the destructor
   7. Visitor
2. **Directed Tests, Corner Cases and Tests from Implementation**.
   1. Empty Tree
   2. Tree with a Single Node
   3. Creating all scenarios for Tree rotation, color shifts and root splits – enumerate & elaborate from the Red-Black Tree algorithms
   4. Multiple Trees within the same scope
      1. Disjoint Lifetime
      2. Overlapped Lifetime
   5. Negative Tests
      1. No-Copy Test – the Tree cannot be copied
      2. No-Assignment Test – the Tree cannot be assigned.
   6. Big Tests.
      1. Tree arising from long (~500) monotonic sequence of insertions
         * Increasing Case
         * Decreasing Case
      2. Tree arising from long alternating sequences of insertions
3. **Use-Model / Extension Tests** – if the Tree is specialized by the user.

For example,

* 1. Add a ‘name’ to a Tree.
  2. Change the type of Key in the Data nodes
  3. Change the type of Info in the Data nodes
  4. Allow for multiple Keys by accommodating a resolution on secondary Keys

1. **Random Tests**.
   1. Pseudo-Random Tests – the data for such tests should be generated beforehand using the random generator and stored. The size of every test should be between 10,000 and 1,000,000. Each category should have at least 5 tests.
      1. No-Duplicate Inserts & Exists-only Finds
      2. Free Inserts & Exists-only Finds
      3. No-Duplicate Inserts & Free Finds
      4. Free Inserts & Free Finds
   2. Random Test – these tests will generate random test data every time the test is run. The random runs should be of the order of 100,000.

The test designed as part of the project was Randomized.

A set of global functions was setup for the random tests of Insert / Find on the

Red Black Tree.

The functions are listed below:

* + 1. int GenerateKey(int iRandRange);

Generates a random Key within the Range for randomization given by iRandRange

2. int GenerateInstruction();

Generates a random Instruction (Insert / Find) on the Red-Black Tree for the random Key

3. void RandomTester(int iRandRange, int iCountData);

The tester that runs for iCountData tests having Keys within the iRandRange

A number of inputs were given say 100, 1000, 10000, 40000 and finally after testing for 1,00,000 inputs (both positive and negative inputs were considered). Validation for all the inputs were done by calling the method Validate() which validates the Tree for BST & Red-Black Conditions.

#### Hash Table implementation

Various scenarios have been enumerated here.

1. **Functional Tests** – test for the primary functionality of the data type.
   1. Create / Destroy Tests – these are for testing various constructors & the destructor.
      1. Automatic Hash Table
      2. Dynamic Hash Table
   2. Insertion Tests
      1. Normal (Common) Cases – without duplicates
      2. Normal (Common) Cases – with duplicates (at least 5 in each case)
   3. Find Tests
      1. Existing Keys (5 cases) – unique
      2. Existing Keys (5 cases) – duplicate / multiplicate
      3. Missing Keys (5 cases)
   4. Delete Tests
2. **API Tests** – test all APIs for the data type (library). All variants for an API (depending on default parameters) need to be tested.
   1. Create – wraps the constructor
   2. Insert
   3. Find
   4. Delete
   5. Dump – should print all keys in the Hash Table
   6. Release – wraps the destructor
   7. Resize
3. **Directed Tests, Collision Cases and Tests from Implementation**.
   1. Empty Hash Table
   2. Hash Table with a Single Node
   3. All cases for Coalesced Chaining.
   4. Multiple Hash Tables within the same scope
      1. Disjoint Lifetime
      2. Overlapped Lifetime
   5. Negative Tests
      1. No-Copy Test – the Hash Table cannot be copied
      2. No-Assignment Test – the Hash Table cannot be assigned.
4. **Use-Model / Extension Tests** – if the Hash Table is specialized by the user. For example,
   1. Add a ‘name’ to a Hash Table.
   2. Change the type of Key in the Data nodes
   3. Change the type of Info in the Data nodes
   4. Change the Hash Function
5. **Random & Huge Tests**.
   1. Pseudo-Random Tests – the data for such tests should be generated beforehand using the random generator and stored. The size of every test should be between 10,000 and 1,000,000. Each category should have at least 5 tests. Use integer keys.
      1. No-Duplicate Inserts & Exists-only Finds
      2. Free Inserts & Exists-only Finds
      3. No-Duplicate Inserts & Free Finds
      4. Free Inserts & Free Finds
   2. Pseudo-Random Tests – repeat for String keys
   3. Random Test – these tests will generate random test data every time the test is run. The random runs should be of the order of 100,000. Use integer keys.

The test designed as part of the project was Randomized. Some were Interactive though. The Random Tests were developed in the same way as that of the Red Black Tree.

The functions are listed below:

1. int GenerateKey(int iRandRange);

Generates a random Key within the Range for randomization given by iRandRange

2. int GenerateInstruction();

Generates a random Instruction (Insert / Find) on the Hash Table for the random Key

3. void RandomTester(int iRandRange, int iCountData);

The tester that runs for iCountData tests having Keys within the iRandRange

The various Test Functions are enumerated:

1. TestData() – Interactive test for the Data Class
2. TestHashNode() - Interactive test for the HashNode Class
3. TestHashTable() - Interactive test for the HashTable Class
4. TestHashTable\_auto() – Automatic test for the HashTable Class
5. TestHashTable\_dyn() – Dynamic test for the HashTable Class
6. TestHashTable\_rand() – Random test for the HashTable Class
7. TestSymbolTable\_rand() - Random test for the SymbolTable Class