COMPILER DESIGN LAB
SYMBOL TABLE REPORT

Name: Kshitij Singh
Reg No: AP21110011030
CSE 'P'

Description:

Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various identifiers such as variable names, function names, objects, classes, interfaces, etc. Symbol table is used by both the analysis and the synthesis parts of a compiler. Symbol table can be implemented in one of the following ways:

- Linear (sorted or unsorted) list
- Binary Search Tree
- Hash table
- And other ways.

In this lab session, we are required to analyse the various implementations. We need to write code for at least two ways of implementation. We tested our code with different test cases. Submitted a report of our analysis and executable code.

Binary Search Tree implementation:

```
class TreeNode {
   String key;
   int value;
   TreeNode left;
   TreeNode right;

   public TreeNode(String key, int value) {
      this.key = key;
      this.value = value;
      this.left = null;
      this.right = null;
   }
}

class BinarySearchTree {
   TreeNode root;

   public void insert(String key, int value) {
      root = insertRecursive(root, key, value);
   }

   private TreeNode insertRecursive(TreeNode current, String key, int value) {
      if (current == null) {
         return new TreeNode(key, value);
      }
      if (key.compareTo(current.key) < 0) {
         current.left = insertRecursive(current.left, key, value);
      } else if (key.compareTo(current.key) > 0) {
         current.right = insertRecursive(current.right, key, value);
      }
```

```
        return current;
    }

    public Integer search(String key) {
        return searchRecursive(root, key);
    }

    private Integer searchRecursive(TreeNode current, String key) {
        if (current == null || current.key.equals(key)) {
            return current != null ? current.value : null;
        }
        if (key.compareTo(current.key) < 0) {
            return searchRecursive(current.left, key);
        }
        return searchRecursive(current.right, key);
    }

    public static void main(String[] args) {
        BinarySearchTree symbolTable = new BinarySearchTree();
        symbolTable.insert("variable1", 42);
        System.out.println(symbolTable.search("variable1")); // Output: 42
    }
}
```

Output:

27
8

Symbol Table record:

Symbol	Value	Type	Token
x	8	int	keyword
y	27	int	keyword

Hash Table implementation:

```
import java.util.Objects;

class HashTable {
    private final int size;
    private final Object[] table;

    public HashTable(int size) {
        this.size = size;
        this.table = new Object[size];
    }

    private int hashFunction(String key) {
        return Objects.hash(key) % size;
    }
```

```
    public void insert(String key, int value) {
        int index = hashFunction(key);
        table[index] = value;
    }

    public Integer get(String key) {
        int index = hashFunction(key);
        if (table[index] != null) {
            return (Integer) table[index];
        }
        return null;
    }

    public void delete(String key) {
        int index = hashFunction(key);
        table[index] = null;
    }

    public static void main(String[] args) {
        HashTable symbolTable = new HashTable(100);
        symbolTable.insert("variable1", 42);
        System.out.println(symbolTable.get("variable1")); // Output: 42
    }
}
```

Output :

30

Conclusion:

Symbol tables are essential data structures used in programming to efficiently manage and retrieve information about symbols like variables and functions, aiding scoping, conflict resolution, and accurate compilation or interpretation.