COMPILER DESIGN LAB
SYMBOL TABLE REPORT

Name: Niteesh Kumar Pandey
Reg No: AP21110011024
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, you are required to analyse the various implementations. You need to write code for at least two ways of implementation. Test your code with different test cases. Submit a report of your analysis and executable code by the end of the session.

Binary Search Tree implementation:

```
import java.util.Scanner;

public class BinarySearchTree {
   public static void main(String[] args) {
      int n, i = 0, j = 0;
      char[] id_Array2 = new char[20];
      char ch;

      System.out.println("Input the expression ending with ; sign:");
      Scanner sc = new Scanner(System.in);
      String input = sc.next();

      while (i < input.length() && input.charAt(i) != ';') {
         id_Array2[i] = input.charAt(i);
         i++;
      }
      n = i - 1;
      System.out.println("\nSymbol Table display");
      System.out.println("Symbol \t addr \t\t\t type");

      while (j <= n) {
         ch = id_Array2[j];

         if (Character.isLetter(ch)) {
            System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t identifier");
            j++;
         } else {
            if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '%' || ch == '=' || ch == '<' || ch == '>') {
               System.out.println("\n " + ch + " \t " + System.identityHashCode(ch) + " \t operator");
               j++;
            }
         }
```

```
    }
  }
}
```

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", 30);
        System.out.println(symbolTable.get("variable1"));
    }
}
```

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.